# NAVAL POSTGRADUATE SCHOOL
# Monterey, California

# THESIS

**DECIMATION OF ENCODING ERRORS IN AN OPTIMUM SNS 2μ LOW-NOISE CMOS ADC**

by

Jeffrey L. Schafer

March 1995

| | |
|---|---|
| Thesis Advisor: | Phillip E. Pace |
| Thesis Co-Advisor: | Douglas J. Fouts |

| REPORT DOCUMENTATION PAGE | Form Approved OMB No. 0704 |
|---|---|

| | | |
|---|---|---|
| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE March 1995 | 3. REPORT TYPE AND DATES COVERED Master's Thesis |

| | |
|---|---|
| 4. TITLE AND SUBTITLE DECIMATION OF ENCODING ERRORS IN AN OPTIMUM SNS 2μ LOW-NOISE CMOS ADC | 5. FUNDING NUMBERS |
| 6. AUTHOR(S) Schafer, Jeffrey L. | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000 | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| | |
|---|---|
| 12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited | 12b. DISTRIBUTION CODE |

13. ABSTRACT *(maximum 200 words)*

Significant research in high performance analog-to-digital converters (ADCs) has been directed at retaining part of the high-speed flash ADC architecture, while reducing the total number of comparators in the circuit. The symmetrical number system (SNS) can be used to preprocess the analog input signal, reducing the number of comparators and thus reducing the chip area and power consumption of the ADC. This thesis examines the issue of encoding errors that result when the separate channels $m_i$ are brought together to derive the input analog voltage. The Very Large Scale Integrated (VLSI) design for the comparators, error checking circuits and Programmable Logic Arrays (PLAs) use the Orbit 2μ CMOS N-well double-metal, double-poly fabrication process. Steady state transfer functions are shown which detail encoding errors that occur when the folded input samples lie at one of the code transition points. To discard the encoding errors that occur, a decimation band is constructed at each transition point. The effectiveness of the decimation band in eliminating the encoding errors and the linearity error is quantified. An Application Specific Integrated Circuit (ASIC) is designed.

| | |
|---|---|
| 14. SUBJECT TERMS Analog-to-Digital Converter; Symmetrical number system; Analog preprocessing foranalog-to-digital conversion; Digital processing for analog-to-digital conversion; VLSI (very large scale integration); MAGIC; CMOS | 15. NUMBER OF PAGES 202 |
| | 16. PRICE CODE |

| | | | |
|---|---|---|---|
| 17. SECURITY CLASSIFICATION OF REPORT Unclassified | 18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified | 20. LIMITATION OF ABSTRACT UL |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

# DECIMATION OF ENCODING ERRORS IN AN OPTIMUM SNS 2μ LOW-NOISE CMOS ADC

Jeffrey L. Schafer
Lieutenant, United States Navy
B.S. Education (Mathematics), Auburn University, 1988

Submitted in partial fulfillment
of the requirements for the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**
**March 1995**

Author: Jeffrey L Schafer
Jeffrey L. Schafer

Approved by: [illegible]
Phillip E. Pace, Thesis Advisor

Douglas J. Fouts
Douglas J. Fouts, Thesis Co-Advisor

Michael A. Morgan
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

# ABSTRACT

Significant research in high performance analog-to-digital converters (ADCs) has been directed at retaining part of the high-speed flash ADC architecture, while reducing the total number of comparators in the circuit. The symmetrical number system (SNS) can be used to preprocess the analog input signal, reducing the number of comparators and thus reducing the chip area and power consumption of the ADC. This thesis examines the issue of encoding errors that result when the separate channels $m_i$ are brought together to derive the input analog voltage. The Very Large Scale Integrated (VLSI) design for the comparators, error checking circuits and Programmable Logic Arrays (PLAs) use the Orbit 2μ CMOS N-well double-metal, double-poly fabrication process. Steady state transfer functions are shown which detail encoding errors that occur when the folded input samples lie at one of the code transition points. To discard the encoding errors that occur, a decimation band is constructed at each transition point. The effectiveness of the decimation band in eliminating the encoding errors and the linearity error is quantified. An Application Specific Integrated Circuit (ASIC) is designed.

## TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

## ACKNOWLEDGMENT

I would first like to say thank you to my wife, Sheryl, for her support of my endeavors and to my daughter, Betty Jo, and my son, Adam for being patient throughout the process of developing and writing this thesis.

I would like to personally thank Lieutenant Commander Richard Carr, USN, for his contributions to this thesis. He is a great friend and inspiring co-worker. His time and assistance provided was well and above the call of duty and contributed greatly.

Finally, I would like to personally thank my thesis advisor, Professor Phillip Pace and coadvisor, Professor Douglas Fouts for allowing me to work for them. Their guidance and understanding were instrumental in the successful completion of this thesis.

# I. INTRODUCTION

## A. FOLDING ANALOG-TO-DIGITAL CONVERTER

Most real-world processes produce analog signals which vary continuously in the time domain. Microprocessors and computers must use binary bit patterns to represent this analog signal which is not easy to store, manipulate or retrieve while using analog technology. Therefore, the need for an analog-to-digital converter (ADC) arises. Analog-to-digital converters are key elements of any system that use digital techniques to process or communicate analog electrical data. There are generally three types of ADCs. The first type is the *slow integrating* ADC, which is good for slowly varying dc voltages. The second type is the successive *approximation* ADC, which can be used to digitize audio signals. The final type is the *flash* ADC, which can digitize video signals and is the most costly. [Ref. 1]

The need arises for ADCs with higher resolution, faster conversion speeds and lower power dissipation. The most popular type of converter with high conversion rate is the *flash* ADC. This architecture requires $2^N - 1$ comparators to achieve $N$-bit resolution. The large number of comparators makes it difficult to keep the power consumption and die area to a minimum. However, the symmetrical number system (SNS) can be used to preprocess the analog input signal, thus reducing the number of comparators, die area and power consumption of the converter.

This thesis presents a folding ADC architecture that incorporates a SNS encoding that has been previously described [Ref. 2]. This scheme improves the resolution and covers a large bandwidth without incurring the power and die area penalties inherent with the *flash* ADC [Ref. 3,4]. The SNS preprocessing scheme is used to decompose the amplitude analyzer operation into a number of parallel sub-operations (moduli) which are of smaller computational complexity. Each moduli symmetrically folds the analog signal with folding period equal to twice the modulus. A small comparator ladder mid-level quantizes each folded output. Each moduli requires a precision in accordance with that modulus. A much higher resolution is achieved after the *N* different SNS moduli are used and the results of these low precision sub-operations are recombined [Ref. 5,6]. The parallel use of folding circuits increases the ADC resolution without increasing the folding rate of the system.

## B. PRINCIPAL CONTRIBUTIONS

Current research has been concentrated on the reduction of die area and power consumption. Research has been on-going to reduce the number of comparators in an ADC which will inherently reduce the die area and power dissipation problem. One method of reducing the number of comparators required is by using a SNS preprocessing scheme. Since the SNS folding waveform is symmetrical, ambiguities exist within each folding period or modulus. Consequently, the dynamic range of the SNS preprocessing depends on the SNS definition and the manner in which the sub-operation (or channels) are recombined. This thesis examines a SNS definition that considerably extends the dynamic range of the SNS folding ADC architecture. Steady state transfer functions are shown which detail encoding errors that occur when the folded input samples lie at one of the code transition points. To discard the encoding errors that occur, a *decimation band* is constructed around each transition point. The effectiveness of the decimation band in eliminating the encoding errors is also examined.

This thesis primarily focuses on the decimation of encoding errors in an optimum SNS 2μ Low-Noise Complementary Metal-Oxide Semiconductors (CMOS) folding ADC. Using an existing design [Ref. 7], layout of the SNS folding circuit and comparators are accomplished utilizing the Very Large Scale Integration (VLSI) Computer Aided Design (CAD) tool, Magic, developed by the University of California at Berkeley. The major focus of this research has been in the inclusion of an error detection circuit which discards the encoding errors that occur when the folded input samples lie at one of the transition points.

Simulation of the circuit layout is conducted using Meta-Software's industrial grade circuit analysis product, HSPICE, in order to verify proper functional operation, to confirm proper layout prior to fabrication and examine the effectiveness of the decimation band in eliminating the encoding errors. An Application Specific Integrated Chip (ASIC) is fabricated using ORBIT's 2μ CMOS N-well double-metal, double-poly process.

## C. THESIS OUTLINE

This thesis is divided into three main parts. The first part (Chapter I-III) discusses the basic concept of an Optimum SNS folding ADC. This includes a design of a unipolar 7-bit SNS ADC using bipolar junction transistor (BJT). The second part (Chapter IV-VI) deals with the CMOS VLSI design, implementation and simulation of an Optimum SNS

9-bit folding ADC. The final part (Chapters VII and VIII) describes the fabrication and further research of the Optimum SNS folding ADC utilizing a 2μ CMOS N-well process.

Chapter II begins with a discussion of the symmetrical number system (SNS) and its application to an analog preprocessing architecture. An overview of the design of an optimum SNS preprocessing 9-bit ADC utilizing this analog preprocessing architecture is presented. This chapter also establishes the software tools used in the layout and simulation of the architecture.

Chapter III introduces a bipolar junction transistor (BJT) model of an Optimum SNS 7-bit folding ADC. The analysis and simulation of a BJT model is examined to demonstrate the optimum SNS preprocessing prior to implementing a CMOS equivalent circuit. Therefore, a unipolar 7- bit folding ADC design with moduli $m_1 = 4$, $m_2 = 5$ and $m_3 = 7$ is examined. The design of the BJT model allows the designer to simulate and analyze the folding ADC on a much smaller scale which reduces the processing time to obtain results and make corrections. Following the basic transient analysis of the folding circuit and comparator group, the decimation of encoding errors are evaluated. The steady state transfer curves are presented for no decimation and for decimation width equal to 25% and 40% of the LSB. Finally, a graph summarizing the encoding errors as a function of decimation width is shown.

Chapter IV deals with the VLSI CMOS architecture of a comparator, latch, parity, and sample and hold circuits. These circuits are presented at this stage because they are easier to visualize in the field effect transistor (FET) form. This will allow the implementation of the CMOS circuits in Magic to go much smoother. Lastly, the generation of the PLA subsystems utilizing the MPLA CAD tool are discussed and truth tables are presented.

Chapter V discusses the implementation of the CMOS VLSI layout for each circuit described in Chapter IV utilizing the Magic CAD tool. A basic FET with emphasis on the MOS transistor is shown with physical and structural design characteristics. Layout design rules are used to obtain a circuit with optimum yield in as small an area as possible without compromising reliability of the circuit. Finally, a system design of the CMOS VLSI folding ADC is presented.

Chapter VI starts with a structured design and simulation approach and its application to CMOS VLSI system design. This is followed by a DC analysis of each modulus comparator group. Simulation of the parity circuit and the PLA subsystem is performed using *esim*. A transient analysis is conducted on the sample and hold circuit. The analysis of the encoding error decimation process is analyzed using an ideal folding waveform as the input to the comparator group. The steady state transfer curves are presented for no decimation and for decimation widths equal to 5%, 10% and 15% of the LSB. Using an existing design of the folding preprocessing circuit, the analysis of the encoding error decimation process is analyzed. The folding waveform is created by ramping the input signal from 0.0 to 3.0 volts. Again, the steady state transfer curves are presented for no decimation and for decimation width equal to 20% of the LSB. A graph summarizing the encoding errors as a function of decimation band width is shown for the ideal folding waveform. Also, a plot illustrating the linearity error is shown. Finally, the dynamic power dissipated in the above CMOS circuits are measured and tabulated using HSPICE.

Chapter VII describes the fabrication process for the system design utilizing ORBIT 2μ CMOS N-well process. A floor plan of the system layout is shown to minimize chip area and maximize speed. Finally the CMOS VLSI implementations of the two chips fabricated for this system design are presented.

Chapter VIII finishes with the conclusion to the thesis. Limitations to the CMOS VLSI design of an ADC are discussed along with recommendations for further research in the use of decimation bands and folding circuits.

## II. BACKGROUND

### A. OPTIMUM SNS

The optimum SNS scheme is composed of a number of *pairwise relatively prime* (PRP) moduli $m_i$. The integers within each SNS modulus are representative of a symmetrically folded waveform with the period of the waveform equal to *twice* the PRP modulus, i.e., $2m_i$. For $m$ given, the integer values within twice the individual modulus is given by the row vector

$$\bar{x}_m = [0, 1, \cdots, m-1, m-1, \cdots, 1, 0]. \tag{1}$$

Figure 1 shows the optimum SNS folding waveforms and SNS output codes for both $m_1 = 4$ and $m_2 = 5$. From (1) the required number of comparators for each channel is $m_i - 1$. Due to the presence of ambiguities, the integers within (1) do not form a complete system of length $2m$ by themselves [Ref. 8]. By recombining the $N$ channels, the SNS is rendered a complete system having a one-to-one correspondence with the residue number system (RNS). For $N$ equal to the number of PRP moduli, the dynamic range $M$ is given by [Ref. 9,10]

$$M = \prod_{i=1}^{N} m_i. \tag{2}$$

This dynamic range is also the position of the first repetitive moduli vector. For example, for $m_1 = 4$ and $m_2 = 3$, the first repetitive moduli vector occurs at an input of 12, as shown in Table 1.

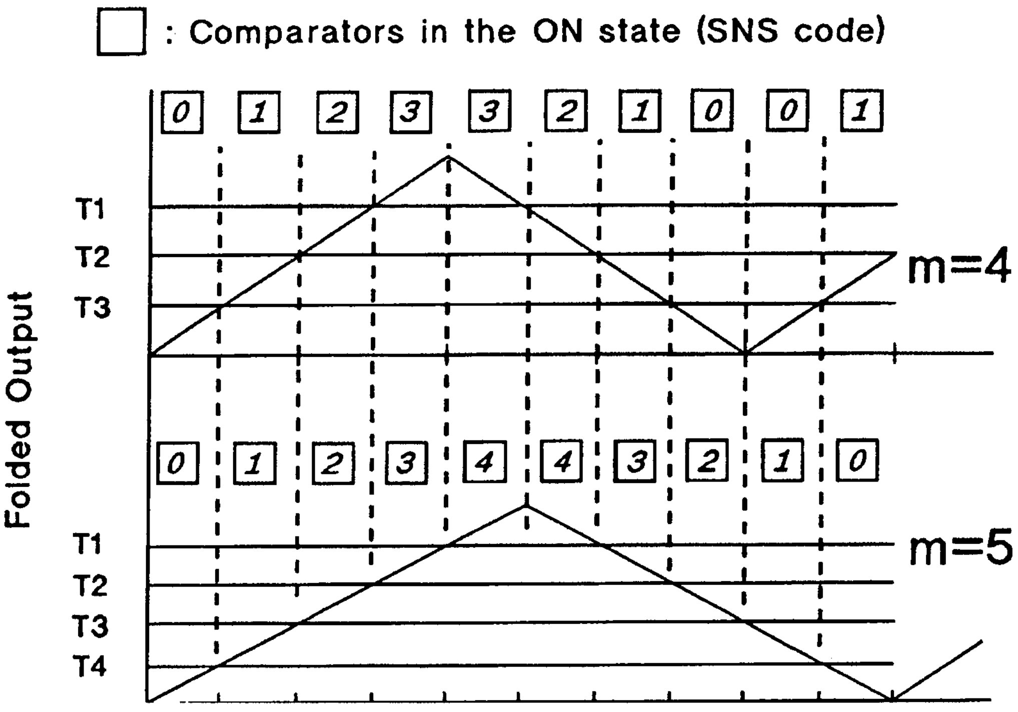


Figure 1. Optimum SNS Folding Waveforms and Output Codes for $m_1 = 4$ and $m_2 = 5$.

## B. PREPROCESSING ISSUES

In an optimum SNS analog preprocessor, a number system is described based on *N* different moduli that will produce the desired dynamic range from (2). An input signal is applied to the *N* different moduli in parallel. Each modulus is defined as a folding circuit that folds the input signal with a period based on twice the value of the modulus. The folded waveform that is produced as the output of each folding circuit is mid-level quantized with a small comparator ladder. The output of the comparator ladder represents the input signal in the SNS format.

| Dynamic Range | SNS Moduli 4 | 3 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 2 |
| 4 | 3 | 1 |
| 5 | 2 | 0 |
| 6 | 1 | 0 |
| 7 | 0 | 1 |
| 8 | 0 | 2 |
| 9 | 1 | 2 |
| 10 | 2 | 1 |
| 11 | 3 | 0 |
| 12 | 3 | 0 |
| | $M = 12$ | |

Table 1. Optimum SNS Preprocessing.

The original design for this thesis incorporated an optimum SNS 9-bit ADC [Ref. 7]. In order to provide 9-bit resolution, a dynamic range $M$ of $2^9$ or 512 is necessary. Moduli chosen to provide this dynamic range $M$ were $m_1 = 7$, $m_2 = 8$, and $m_3 = 11$. Using (2) to calculate the dynamic range provided by the optimum SNS encoding yields a value of 616, which is well above the required 512.

Figure 2 shows a block diagram of a Optimum SNS 9-bit ADC with error correction along with the number of outputs from each major block. The analog input signal is applied to each of the moduli in parallel. Each modulus is composed of a number of folding stages that produce a folded waveform with a period equal to twice the modulus. A small comparator ladder consisting of $m_i$ - 1 comparators mid-level quantizes the folded output from each modulus. With the SNS preprocessing, the modulus must be recombined in e.g. a read-only memory (ROM) in order to resolve the encoded input analog voltage. Consequently, there is a possibility of an encoding error when the input voltage happens to lie at one of the code transition points. The encoding errors occur at the transition points when some comparators at a position to change do change while others do not [Ref. 10]. To eliminate the encoding errors, $2m_i$ comparators (instead of $m_i$-1) are used for the channel with the smallest modulus. For the 9-bit ADC, $2m_i$=14 comparators are required. The comparators are positioned to place a small *decimation band* around each transition point where the folding waveforms cross the threshold levels. If the folded input voltage falls within one of these small bands the number of comparators in the ON state is *even*, otherwise, the number is *odd*. A parity circuit is used to discard

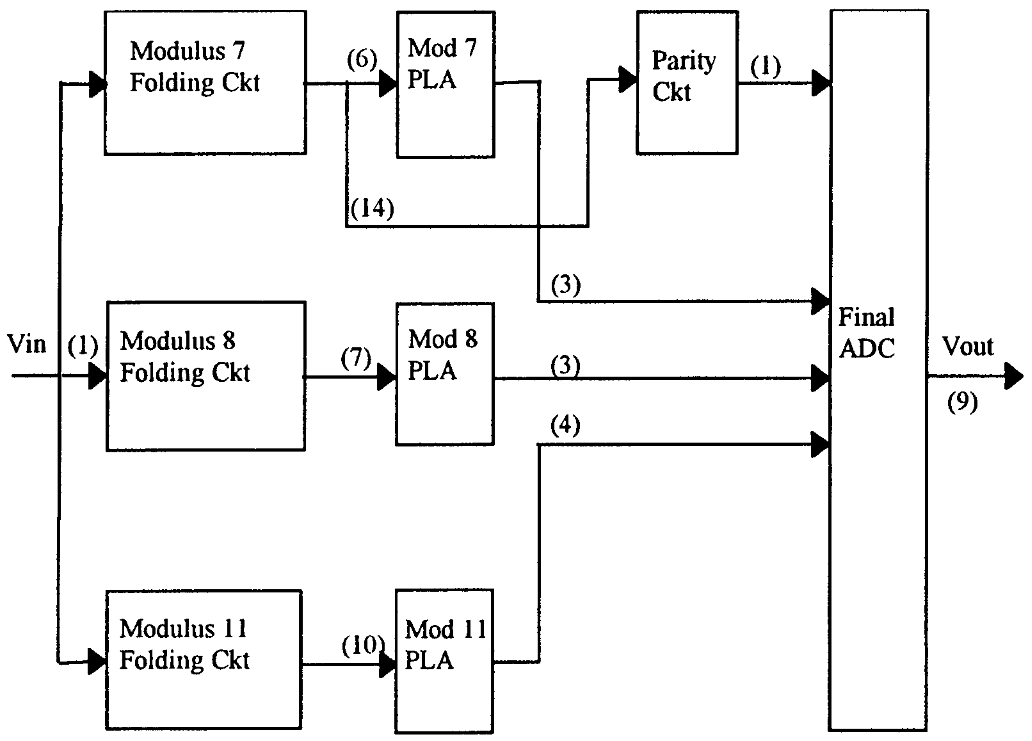


Figure 2. Optimum SNS 9-bit ADC Block Diagram.

the sample if the parity is *even*. If the sample is discarded, the output is latched at the previous known good state. The outputs of the comparator ladders are applied to a modulus programmable logic array (PLA). The output of each modulus PLA represents the thermometer code in its optimum SNS binary format. A final PLA is then used to recombine the channels and transform the encoded input signal into a more common representation (e.g. binary number).

## C. COMPUTER AIDED DESIGN AND SIMULATION TOOLS

### 1. Magic

Magic is an interactive editor used for creation of VLSI circuit layouts. All of the VLSI layouts completed for this thesis were accomplished using this CAD tool. Using a color graphics display and a mouse, the designer can construct basic cell layouts and combine them hierarchically into larger integrated circuits. Magic contains a design rule checker (DRC) that ensures compliance with layout rules for the particular technology being used. As a means of interfacing with other design and simulation programs, Magic allows the designer to extract the developed layout into the native language of other programs. [Ref. 11]

### 2. Ext2spice and Ext2sim

Ext2spice and ext2sim reads a file in the **.ext** format and creates a new file in the **.spice** and **.sim** format respectively. The **.spice** file created contains a list of transistor and capacitor instantiations. The designer must then add the transistor models, in the form of Spice level 2 parameters, and other simulation information in order to produce an executable file in the **.spice** format. This CAD tool assumes ground node is 0, Vdd is node 1, and that node 2 is reserved as an error node. The **.sim** file created is ready for use by *esim*. The *esim* CAD tool assumes that the source voltage and ground are labeled Vdd and GND respectively. [Ref. 11]

### 3. MPLA

MPLA is a Programmable Logic Array (PLA) generator that generates Magic layout compatible PLAs in various styles and technologies. This thesis work is completed in scaleable CMOS cis version (SCS3cis). It supports MOSIS 1.5/2.0/3.0 micron SCMOS process, pseudo-nmos static PLA with p-channel pullups. The *cis* indicates buried contacts with inputs and outputs on the same side of the PLA. The modulus and ADC PLAs are generated using standard inputs and outputs, extra ground lines every 10 rows in the AND logic plane and every 10 columns in the OR logic plane. [Ref. 11]

### 4. Esim

Esim is an event-driven switch level simulator for CMOS transistor circuits. Once the circuit is extracted into a **.sim** format, *esim* is used to watch the various nodes, set or reset nodes, and to simulate the logical operation of the circuit. The nodes can then be inspected, and the circuit evaluated. The modulus and final ADC are simulated using *esim* CAD tool. [Ref. 11]

### 5. HSPICE

HSPICE is an optimizing analog circuit simulator by Meta-Software used for the circuit simulation of the VLSI circuit design. HSPICE is used for the steady-state and transient simulation of the circuit design. HSPICE program is compatible with most SPICE variations, has superior convergence, accurate modeling, and hierarchical node naming and references. Once the layout is complete using Magic the design is extracted using the *ext2spice* CAD tool. The designer must then add the transistor models, voltage sources and simulation information. Note that due to the large number of transistors and capacitors in this design along with the simulation of both analog and digital components, the processing time for the simulation is very long. The Graphical Simulation Interface (GSI) is used to graphically display the waveforms at various nodes in the VLSI design to ensure proper functional operation and design layout. [Ref. 12]

### 6. MATLAB

MATLAB is a high-performance interactive software package for numeric computation which incorporates numerical analysis, signal processing, and graphics into an easy-to-use environment. During the simulation of the VLSI design using HSPICE, various nodes are selected and the voltages from those nodes are collected in a data file. Once the simulation is complete, the data file is converted into a MATLAB m-file and a plot of the transfer curve for a Optimum SNS 9-bit folding ADC is obtained. [Ref. 13]

## 7. MicroSim's Design Center

MicroSim's Design Center CAE system provides an integrated environment to capture, simulate, and analyze analog and digital circuit design. *Pspice* is the analog and digital circuit simulator whereas *Schematics* is the graphical circuit editor used to layout the circuit in the schematic format. The Design Center is used to analyze a Bipolar Junction Transistor (BJT) Optimum SNS 7-bit folding ADC discussed in Chapter II. Once the BJT design is complete using *Schematic Capture*, the netlist (**.net**), alias (**.als**), and circuit (**.cir**) files were created. Using the DOS editor, the **.cir** and **.als** file were modified to create an ASCII data file when *Pspice* is invoked. Shown in Appendix A are the modifications (italicized) required to the **.cir** and **.als** files to create ASCII data vice binary data file for the selected output nodes. [Ref. 14]

# III. BI-POLAR JUNCTION TRANSISTOR (BJT) ARCHITECTURE

To demonstrate the optimum SNS preprocessing and the decimation of encoding errors, a unipolar 7-bit ADC with $m_1 = 4$, $m_2 = 5$, and $m_3 = 7$ is considered. For $N = 3$ (number of PRP moduli), the dynamic range $M$ (2) is M = 140. A block diagram of this 7-bit folding ADC is shown in Figure 3.

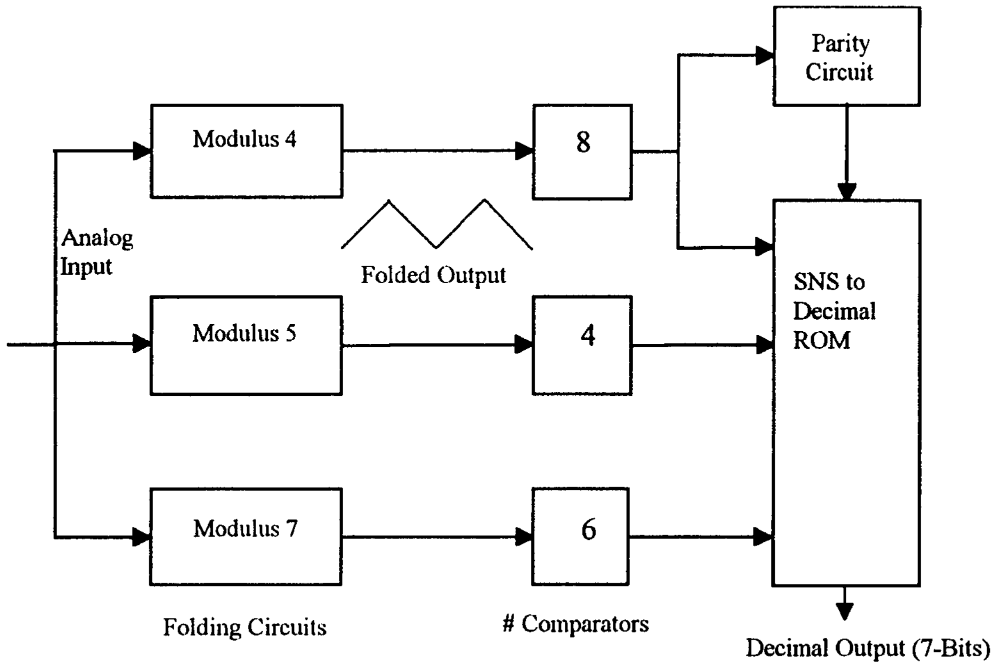


Figure 3. 7-bit SNS Folding ADC Block Diagram.

## A. FOLDING CIRCUIT

The first step in the design of the 7-bit SNS Folding ADC is the design of the individual folding stages. High performance folding circuits that utilize bipolar technology often use several identical but independent stages connected in parallel with one stage for each required fold. Figure 4 shows one stage of a $m_1 = 4$ folding circuit including the three required comparators. The folding stage is composed of a pair of differential

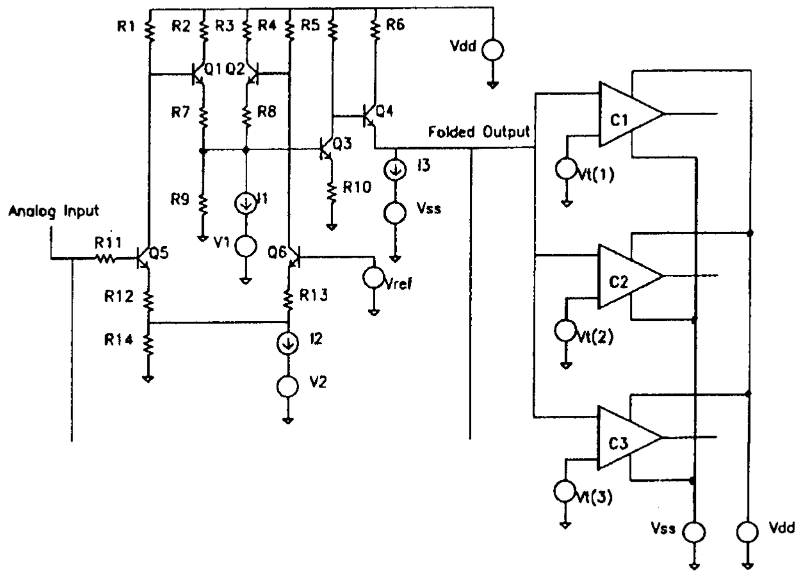


Figure 4. One stage of $m_i = 4$ folding circuit.

amplifiers. The input analog signal is applied to the base of Q5 with the folded output taken at the emitter of Q4. To fold the analog signal correctly at each stage, a reference voltage $V_{ref}$ is supplied to the base of Q6. Transistors Q3 provide some gain and dc voltage level shifting while the emitter follower Q4 is used to ensure low output resistance of the folding stage. The least significant bit (LSB) code width is set to 12mv (full scale voltage $V_{FS} = 2^N$ LSB = 1.536v). The folding period for each modulus contains $2m_i$ LSBs. The folding period for the modulus 4 channel is $T_4 = 2m_i$ LSB = 0.096v. The folding periods for the other two channels are scaled appropriately as $T_5$ = 0.120v and $T_7$ = 0.168v. The first required reference voltage (above zero) is the $V_{ref}(1) = T_4/2$ = 0.048v. Table 2 shows the reference voltages used for each stage of the three folding circuits required.

The height of each folding waveform output from each folding circuit is different. To solve this problem, resistor R2 and R3 are varied to pull the folded waveform up, while R12 and R13 are varied to pull the folded waveform down. This ensured that each folded

| Fold No. | $V_{ref}$ (volts) | | |
|---|---|---|---|
| | $m_1 = 4$ ($T_4 = 0.096$) | $m_2 = 5$ ($T_5 = 0.120$) | $m_3 = 7$ ($T_7 = 0.168$) |
| 0* | -0.048 | -0.060 | -0.084 |
| 1 | 0.048 | 0.060 | 0.084 |
| 2 | 0.144 | 0.180 | 0.252 |
| 3 | 0.240 | 0.300 | 0.420 |
| 4 | 0.336 | 0.420 | 0.588 |
| 5 | 0.432 | 0.540 | 0.756 |
| 6 | 0.528 | 0.660 | 0.924 |
| 7 | 0.624 | 0.780 | 1.092 |
| 8 | 0.720 | 0.900 | 1.260 |
| 9 | 0.816 | 1.020 | 1.428 |
| 10 | 0.912 | 1.140 | 1.596 |
| 11 | 1.008 | 1.260 | - |
| 12 | 1.104 | 1.380 | - |
| 13 | 1.200 | 1.500 | - |
| 14 | 1.296 | - | - |
| 15 | 1.392 | - | - |
| 16 | 1.488 | - | - |

* Calibration folding stage

Table 2. Folding Circuit Reference Voltages ($V_{FS} = 1.536$v).

waveform output is of equal height as shown in Figure 5. Also note that the waveforms are folding at the correct folding period. Table 3 tabulates the component values for the folding circuit of each modulus.

## B. COMPARATOR CIRCUIT

The folded output of each modulus channel is then fed to the corresponding comparator circuits. The total number of comparators required for the 7-bit system is 13 (each channel has $m_i$ - 1 comparators). The comparator threshold voltages $V_t$ are derived from each folding waveform to mid-level quantize the input signal into the SNS format. The matching voltages $V_t(i)$ for the comparator thresholds in each channel are shown in Table 4. Note the threshold values are not uniformly spaced. The steady state folding waveforms and comparator outputs from each channel are shown in Figures 6, 7, and 8 for $m_1 = 4$, $m_2 = 5$ and $m_3 = 7$ respectively. Note that the comparator outputs are turned ON and OFF in a thermometer type format.

## C. DECIMATION OF ENCODING ERRORS

With the optimum SNS preprocessing, the channels must be recombined in a read-only memory (ROM) in order to resolve the encoded input analog voltage. Consequently, there is a possibility of an encoding error when the input voltage happens to lie at one of the code transistion points. The encoding errors occur at the transition points when some comparators at a position to change do change while others do not [Ref. 3]. Figure 9 shows the steady state transfer function for an optimum SNS ADC using a sampling period $\Delta v$=0.4mv. To eliminate the encoding errors, $2m$ comparators (instead of $m$ - 1) are used for the channel with the *smallest* modulus. For the 7-bit system, $2m_1 = 8$ comparators are required (see Figure 3). The comparator threshold voltages are set to place a small *decimation band* around each transition point where the folding waveforms cross the threshold levels. Figure 10 details the placement of the various threshold voltages. If the folded input voltage falls within one of these small bands the number of comparators in the ON state is *even*, otherwise, the number is *odd*. A parity circuit is used

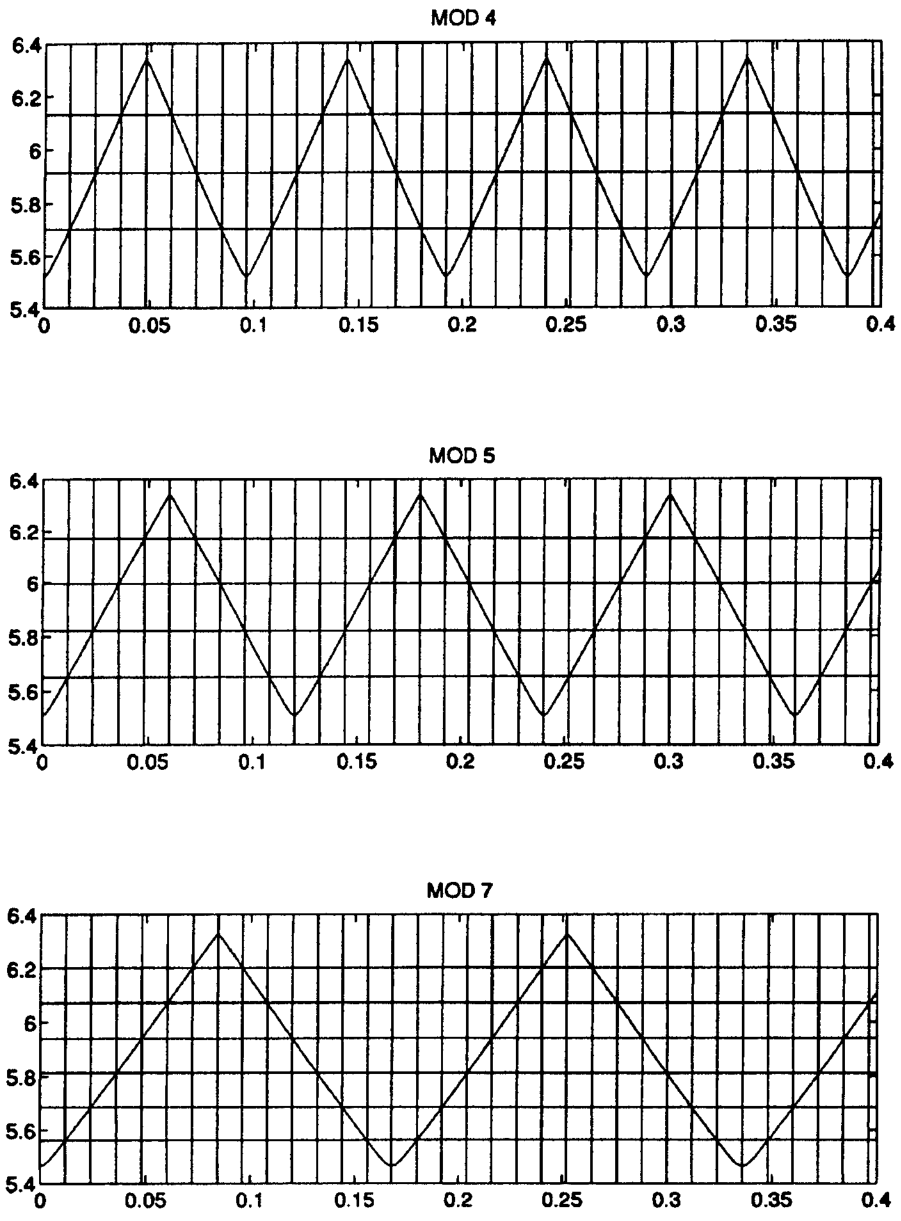


Figure 5. SNS Folding Circuit Outputs.

| Component | $m_1 = 4$ | $m_2 = 5$ | $m_3 = 7$ |
|---|---|---|---|
| R2 | 1.185K | 1.19K | 1.18K |
| R3 | 1.185K | 1.19K | 1.18K |
| R12 | 50 | 75 | 120 |
| R13 | 50 | 75 | 120 |
| Q1, Q2 | beta = 200 | beta = 200 | beta = 200 |
| Q3, Q4 | beta = 100 | beta = 100 | beta = 100 |
| Q5, Q6 | beta = 200 | beta = 200 | beta = 200 |

Table 3. Component Values for Folding Circuit.

| Comparator No. | $V_t$ (volts) | | |
|---|---|---|---|
| | $m_1 = 4$ | $m_2 = 5$ | $m_3 = 7$ |
| 1 | 5.7030 | 5.6523 | 5.5610 |
| 2 | 5.9140 | 5.8203 | 5.6850 |
| 3 | 6.1330 | 5.9945 | 5.8120 |
| 4 | - | 6.1716 | 5.9410 |
| 5 | - | - | 6.0720 |
| 6 | - | - | 6.2030 |

Table 4. Comparator Threshold Voltages (BJT).

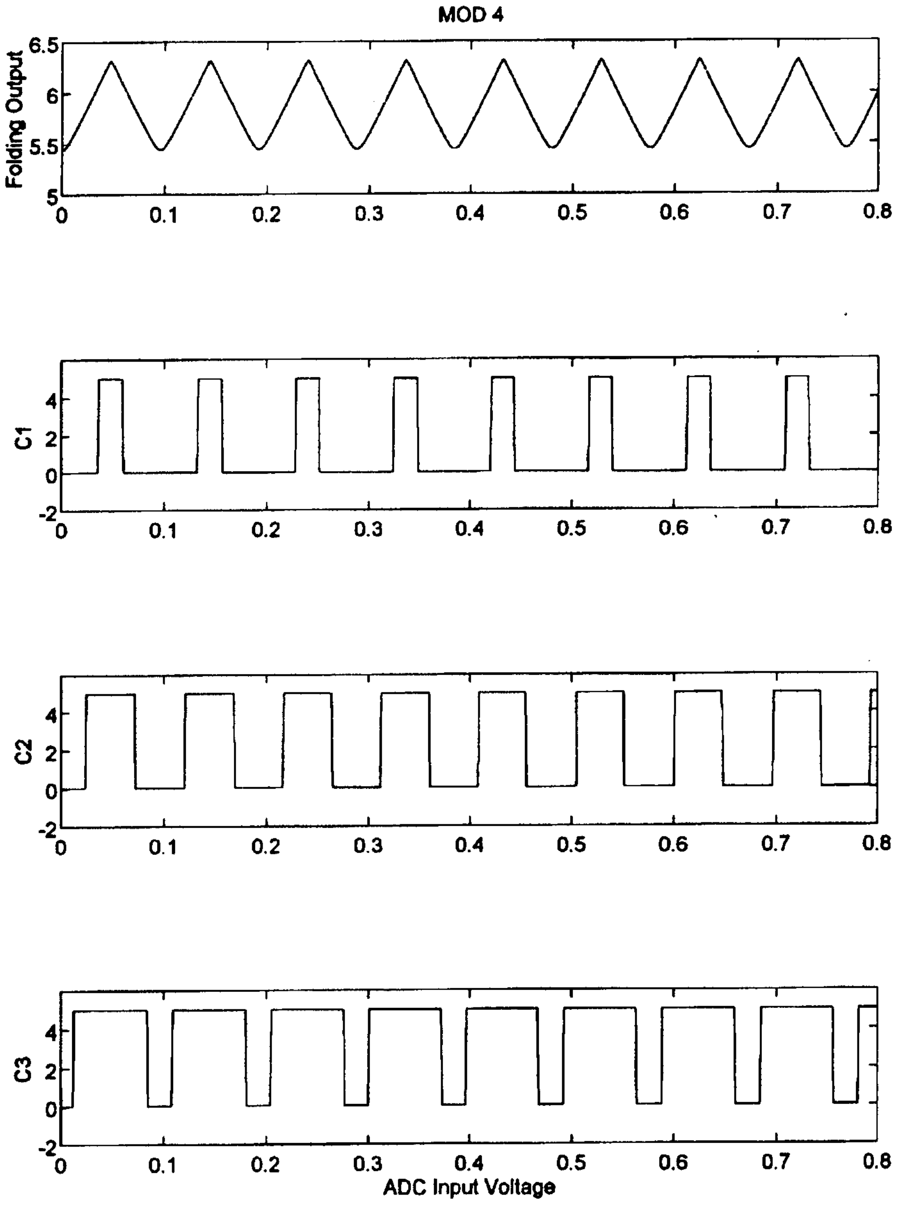


Figure 6. Folding Waveform and Comparator Outputs for $m_1 = 4$.

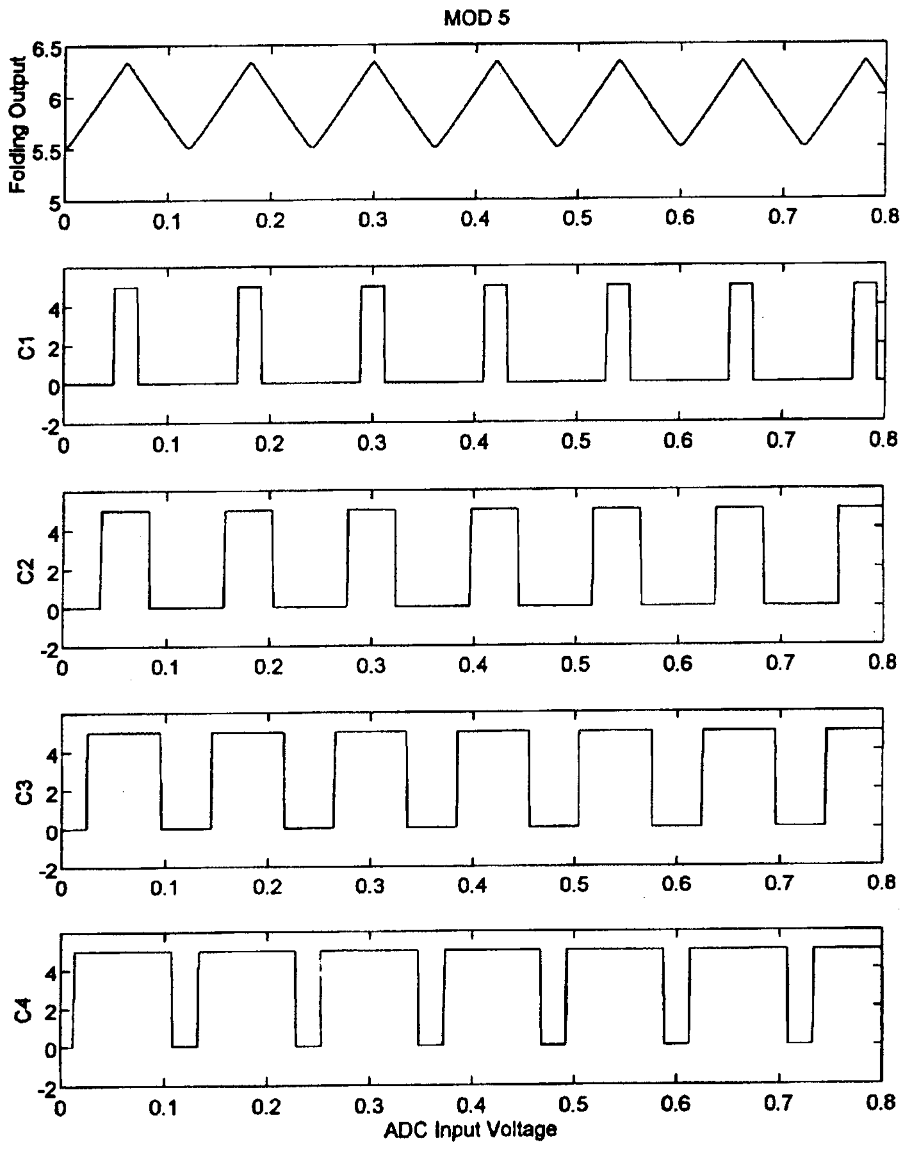


Figure 7. Folding Waveform and Comparator Outputs for $m_2 = 5$.

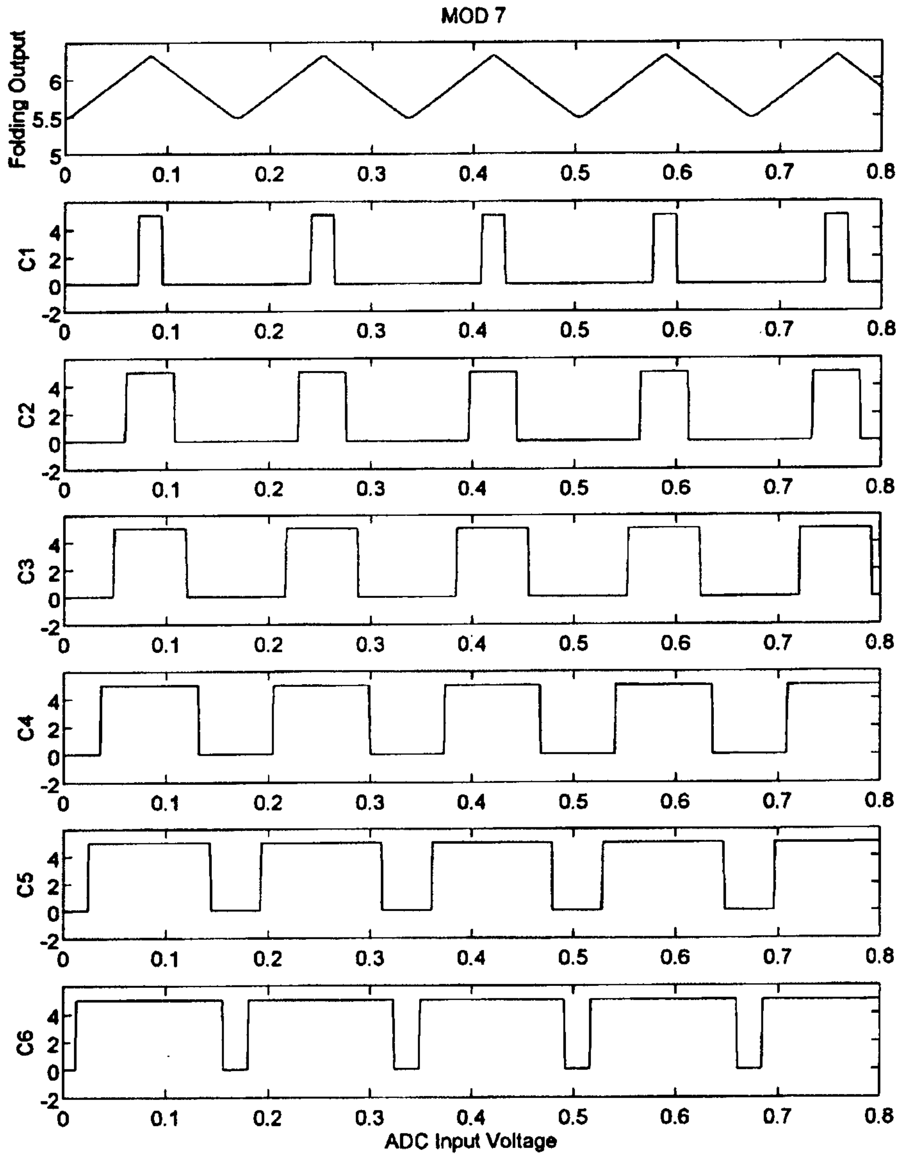


Figure 8. Folding Waveform and Comparator Outputs for $m_3 = 7$.

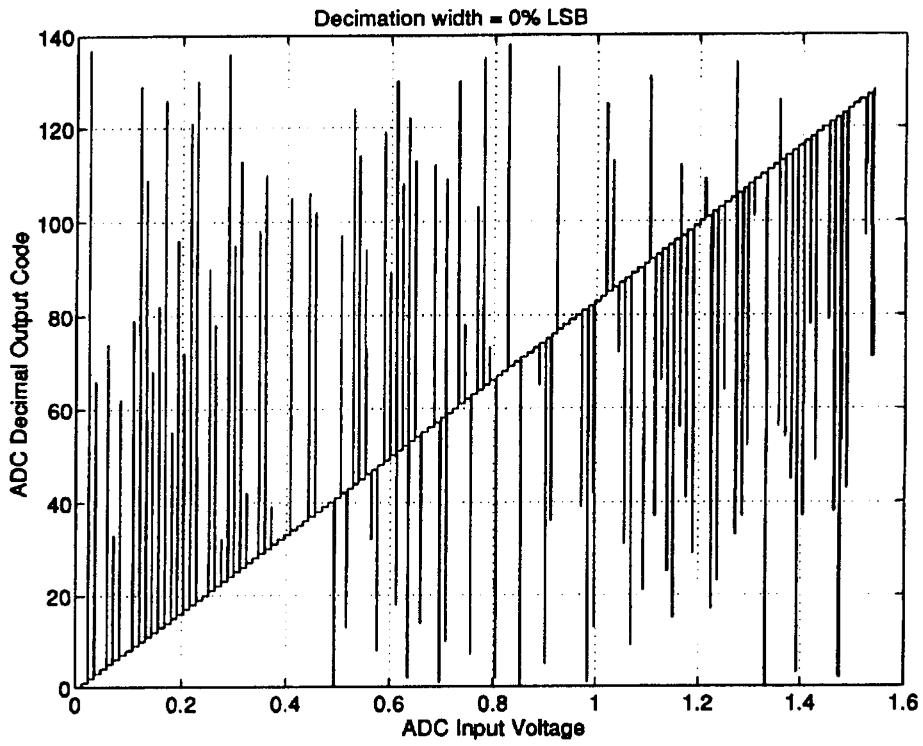


Figure 9. Steady State Transfer Function.

to discard the sample if the parity is *even*. If the sample is discarded, the output is latched at the previous known good state.

To quantify the effectiveness of the decimation band in eliminating the encoding errors, the width is varied from 0% of the LSB code width (no decimation) to 40% of the LSB code width. Figure 11 shows the transfer function for a decimation width equal to 25% of the LSB code width. Note the reduction in the number of errors from the transfer function in Figure 9. Figure 12 shows the transfer function for a decimation width of 40%

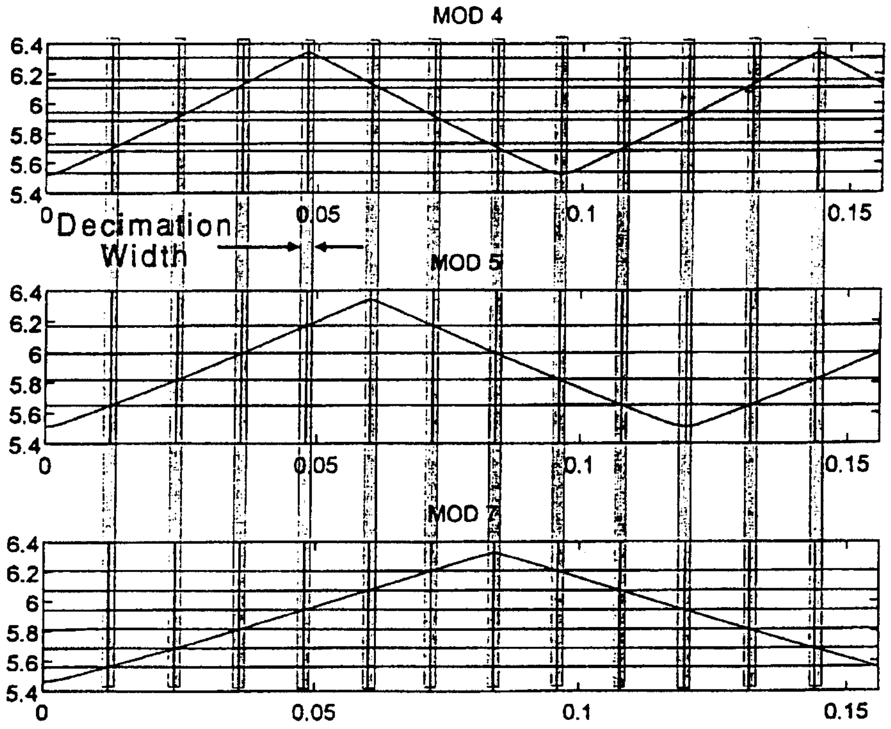


Figure 10. Decimation Bands at the Code Transition ~~Points.~~

of the LSB code width. At this width the encoding errors are almost completely removed. The performance of the decimation band is summarized in Figure 13. As the decimation increases, the number of encoding errors decrease. The corresponding matching threshold values for the decimation band comparators ($m_1$ = 4 channel) are given in Table 5. It is important to note in this example that the performance of the decimation band is a reflection on how well the crossing points are lined up across each channel. Lining up these transition points more accurately can further reduce the required decimation width.

Appendix B is the MATLAB m-file code written to simulate the SNS to decimal ROM (see Figure 3). The ASCII data obtained from the comparator outputs is used to derive the 7-bit binary output code.

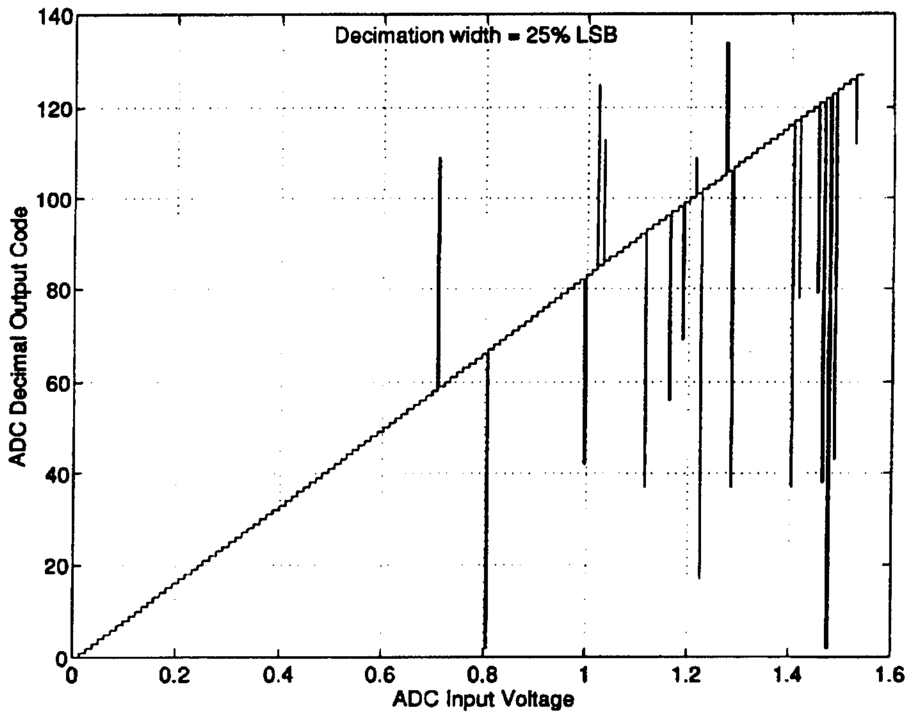


Figure 11. Steady State Transfer Function.
(Decimation Width = 25% LSB)

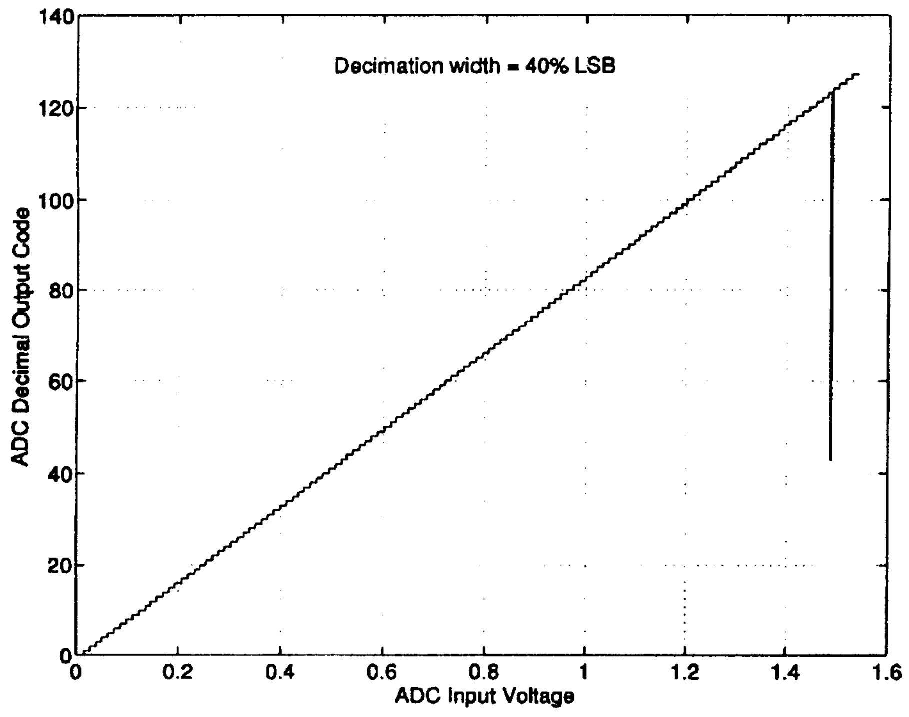


Figure 12. Steady State Transfer Function.

(Decimation Width = 40% LSB)

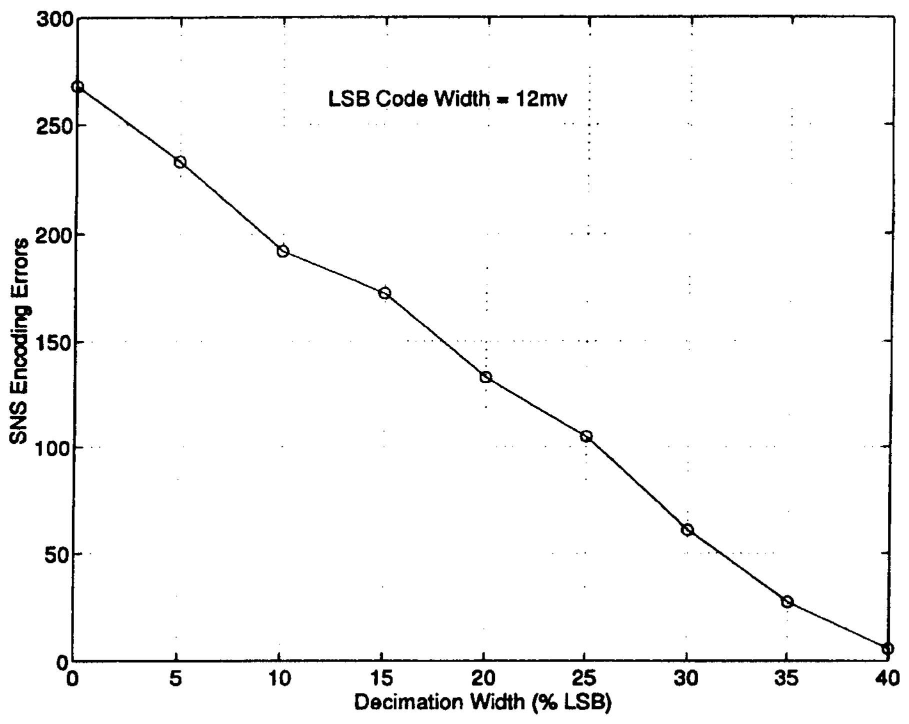


Figure 13. Number of Encoding Errors as a Function of Decimation Width.

| Comp. No. $m_1 = 4$ | $V_t$ (volts) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | % of LSB 5 | % of LSB 10 | % of LSB 15 | % of LSB 20 | % of LSB 25 | % of LSB 30 | % of LSB 35 | % of LSB 40 |
| 1 | 5.526 | 5.527 | 5.528 | 5.529 | 5.530 | 5.531 | 5.532 | 5.534 |
| 2 | 5.700 | 5.693 | 5.688 | 5.683 | 5.678 | 5.673 | 5.668 | 5.663 |
| 3 | 5.706 | 5.713 | 5.718 | 5.723 | 5.728 | 5.733 | 5.738 | 5.743 |
| 4 | 5.911 | 5.904 | 5.898 | 5.893 | 5.888 | 5.883 | 5.877 | 5.872 |
| 5 | 5.917 | 5.924 | 5.930 | 5.935 | 5.940 | 5.946 | 5.951 | 5.956 |
| 6 | 6.130 | 6.122 | 6.117 | 6.112 | 6.106 | 6.101 | 6.096 | 6.090 |
| 7 | 6.137 | 6.144 | 6.149 | 6.154 | 6.160 | 6.165 | 6.170 | 6.176 |
| 8 | 6.305 | 6.304 | 6.303 | 6.302 | 6.301 | 6.300 | 6.298 | 6.297 |

Table 5. Decimation Band Comparator Threshold Voltages (BJT).

# IV. VLSI ARCHITECTURE

Figure 14 shows an optimum SNS 9-bit folding ADC that is being investigated in a VLSI architecture [Ref. 15]. The three channels consist of $m_1 = 7$, $m_2 = 8$ and $m_3 = 11$. The output from each channel is quantized with $m_i$ - 1 comparators. In order to resolve the encoded input analog voltage the comparator outputs must be recombined through a programmable logic array (PLA). This chapter will investigate the basic comparator, latch, parity circuit, output latch circuit and PLA subsystem design and how they work together to eliminate encoding errors.

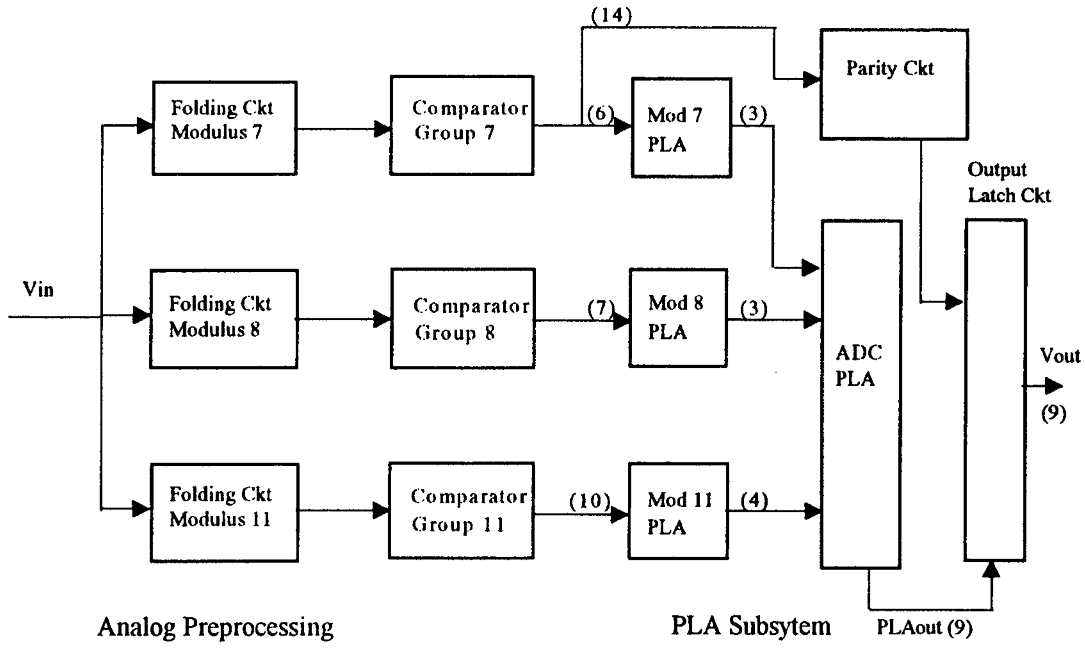


Figure 14. Block Diagram of the 9-Bit SNS Folding ADC.

## A. COMPARATOR CIRCUIT

The comparator circuit is designed to turn ON when the folded waveform reaches a particular threshold voltage. Figure 15 shows the basic comparator circuit used. The circuit is composed of a CMOS differential amplifier, M1 and M2. The folded waveform input is applied to the gate of M1, while a threshold voltage $V_t$ is applied to the gate of M2. Transistors M5 is a current source and provides proper biasing to the differential amplifier. Transistors M3 and M4 are active loads. The output of the differential amplifier is taken from the drain of M2 and applied to the gates of transistors M6 and M7. These two transistors form an inverter which ensures the output is compatible with CMOS logic gates. When the folded input waveform applied to the gate of M1 reaches the

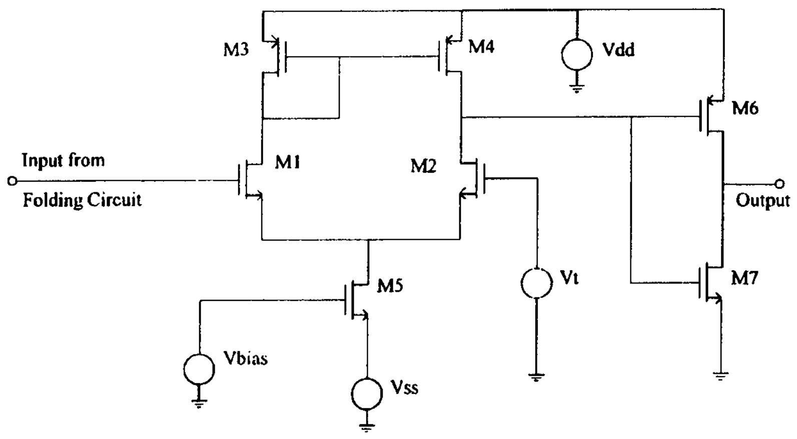


Figure 15. Comparator Circuit.

threshold voltage applied to the gate of M2 the comparator output goes low (0v), otherwise the output is high (+5v). The matching voltages $V_t(i)$ for the comparator thresholds in each channel are shown in Table 6.

Once the comparator is simulated correctly, the appropriate number of these comparators are connected in parallel to the modulus folding circuit. The total number of comparators required for the 9-bit ADC is 23 (each channel has $m_i$ - 1 comparators). To accommodate for the elimination of encoding errors, $2m_i$ comparators are used for the

| Comparator Number | $V_t$ (volts) | | |
|---|---|---|---|
| | $m_1 = 7$ | $m_3 = 8$ | $m_3 = 11$ |
| 1 | 2.5300 | 2.5600 | 2.6016 |
| 2 | 2.2154 | 2.3600 | 2.5021 |
| 3 | 1.9085 | 2.0007 | 2.2026 |
| 4 | 1.6200 | 1.7278 | 2.0031 |
| 5 | 1.2947 | 1.4549 | 1.8036 |
| 6 | 0.9878 | 1.1820 | 1.6041 |
| 7 | - | 0.9091 | 1.4046 |
| 8 | - | - | 1.2051 |
| 9 | - | - | 1.0056 |
| 10 | - | - | 0.8061 |

Table 6. Comparator Threshold Voltages (VLSI).

channel with the smallest modulus. For the 9-bit ADC, $2m_1 = 14$ comparators are required (see Figure 14). The corresponding matching threshold voltages for the decimation band comparators ($m_1 = 7$ channel) are given in Table 7. Note that the threshold voltages listed in Table 6 and 7 are for an ideal folding output waveform.

| Comparator No. | $V_t$ (volts) | | |
|---|---|---|---|
| | 5% LSB | 10% LSB | 15% LSB |
| 1 | 2.8200 | 2.8100 | 2.8000 |
| 2 | 2.5600 | 2.5700 | 2.5800 |
| 3 | 2.5200 | 2.5100 | 2.5000 |
| 4 | 2.2500 | 2.2600 | 2.2700 |
| 5 | 2.2000 | 2.1900 | 2.1800 |
| 6 | 1.9300 | 1.9400 | 1.9500 |
| 7 | 1.8900 | 1.8800 | 1.8700 |
| 8 | 1.6400 | 1.6500 | 1.6600 |
| 9 | 1.5900 | 1.5800 | 1.5700 |
| 10 | 1.3300 | 1.3400 | 1.3500 |
| 11 | 1.2700 | 1.2600 | 1.2550 |
| 12 | 1.0200 | 1.0300 | 1.0400 |
| 13 | 0.9800 | 0.9700 | 0.9600 |
| 14 | 0.7000 | 0.7100 | 0.7200 |

Table 7. Decimation Band Comparator Threshold Voltages (VLSI).

## B. LATCH CIRCUIT

The original design of the latch circuit is shown in Figure 16 [Ref. 4]. This simple circuit consisted of a transmission gate followed by a capacitor to serve as the latch circuit prior to the modulus PLAs. Though this circuit performed well in simulation it is found to be unsatisfactory in the VLSI design due to the size of the capacitor (5nF). The real estate required for this size of capacitor would be upwards of 1 $mm^2$.

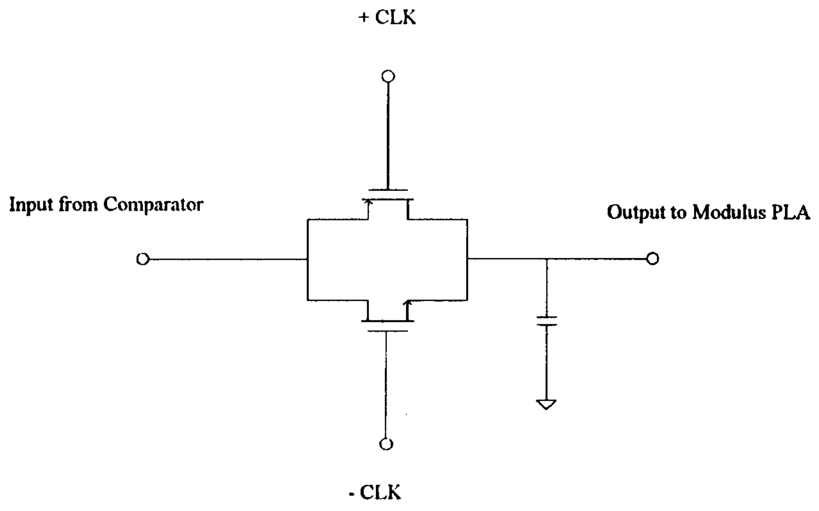


Figure 16. Original Sample and Hold Circuit.

Figure 17 shows the new design of the latch circuit used at the output of the comparators. By cascading a transmission gate with an inverter the tristate inverter is constructed, transistors M1 through M4 and M7 through M10. Transistors M5 and M6 form the buffering input inverter. The latch circuit is controlled by two non-overlapping

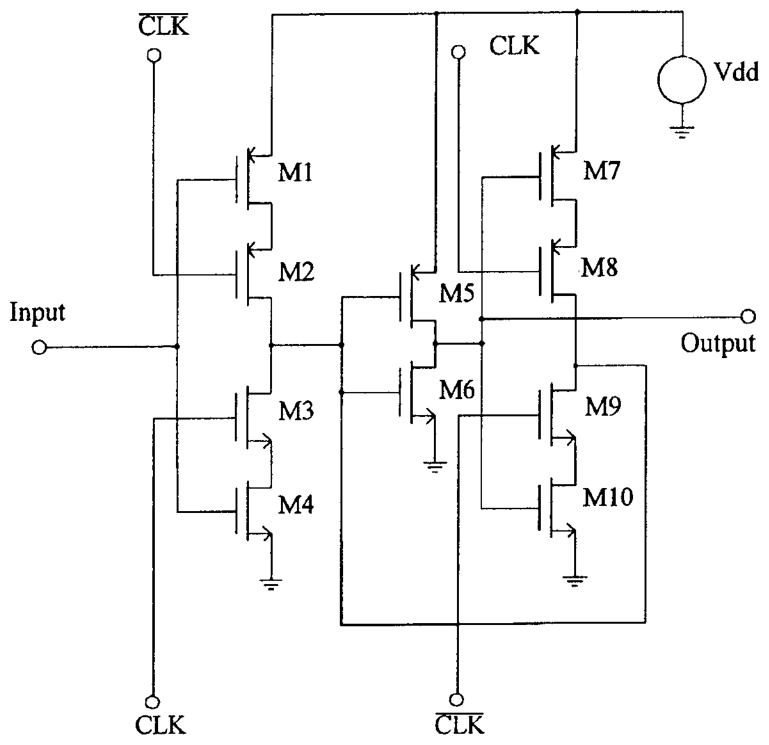


Figure 17. Latch Circuit.

clocks, CLK and $\overline{\text{CLK}}$. When CLK = 1 and $\overline{\text{CLK}}$ = 0, the input signal is coupled through the first tristate inverter (M1 through M4) into the buffer input inverter (M5 and M6) and the output of the latch is identical to the input signal. The second tristate inverter (M7 through M10) is in a tristate condition (the output is not driven by the input). However, when CLK = 0 and $\overline{\text{CLK}}$ = 1, the first tristate inverter is in the tristate condition and the input signal is blocked from the buffer inverter. The second tristate inverter is now coupled to the input of the buffer inverter which allows the output of the latch to remain in its previous state until another sample has been taken.

These latches are connected to the comparator circuits discussed in the previous section. The composition of the comparator and latch circuit for each modulus is referred to as the *comparator group*.

## C. PARITY CIRCUIT

The comparator threshold voltages for the smallest channel are set to place a small *decimation band* around each transition point where the folding waveform crosses the threshold level (see Figure 10.). A parity circuit is used to determine if the input folding waveform falls within one of these bands. If so, the number of comparators in the ON state is *even* and the parity circuit will discard that sample.

Figure 18 shows the symbolic layout of an even-parity circuit. That is, the output will be 1 if an even number of inputs are 1. The circuit is made up of a number of 2-input Exclusive OR (XOR) gates connected in a tree-like structure with *N*-inputs and a single output. Figure 19 shows a 2-input XOR gate CMOS circuit used to construct the even-parity circuit [Ref. 16]. Transistors M1 and M2, M3 and M4 form an inverter while transistors M5 and M6 form a transmission gate. When the A input is high, the output of the first inverter (M1 and M2) is low. The transmission gate (M5 and M6) is disabled and the second inverter (M3 and M4) is enabled with the compliment of B input as the output. When the A input is low, the output of the first inverter is high, the second inverter is disabled and the transmission gate is enabled. The B input will be passed to the output. The truth table for a 2-input XOR function and the even-parity circuit is shown in Table 8 and Table 9 respectively. Note that the inputs to Table 9 represents the thermometer format outputs of modulus 7 comparator group.

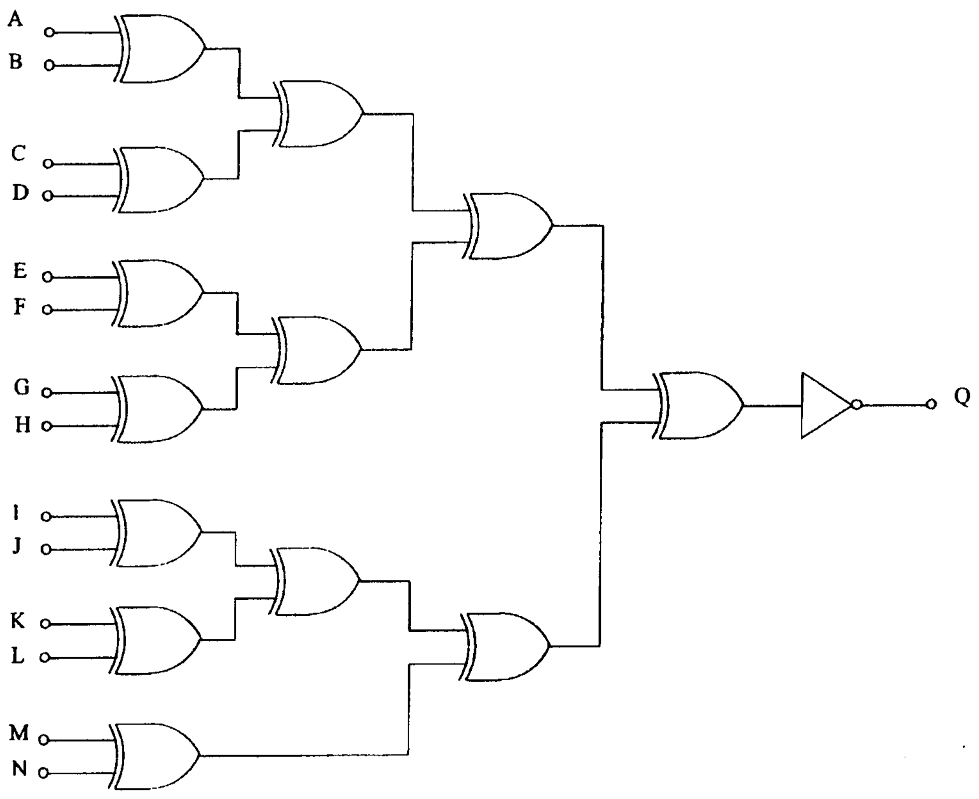


Figure 18. Symbolic Layout of Even-Parity Circuit.

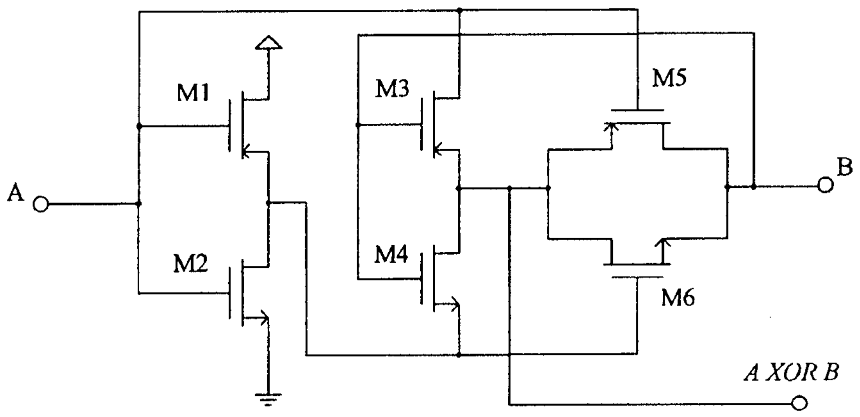


Figure 19. 2-Input XOR CMOS Gate Circuit.

| Input Signal | | Output Signal |
|---|---|---|
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table 8. Truth Table for 2-Input XOR Function.

| Inputs | | | | | | | | | | | | | | Output |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | Q |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 9. Truth Table for Even-Parity Circuit.

## D. OUTPUT LATCH CIRCUIT

In order for the parity circuit to hold or discard a particular input sample, an output latch circuit is designed using D flip-flops. Figure 20 shows the symbolic layout and Figure 21 is the CMOS equivalent circuit. Two non-overlapping control signals are required to properly clock the circuit. The *even-parity circuit* provides both control signals. First, the output of the parity circuit is passed through a buffer to create the S control signal. The output of the parity circuit is also routed through an inverter to form the $\overline{S}$ control signal. This ensures that both control signals (S and $\overline{S}$) are delayed the same amount.

Transistors M1 and M2, M3 and M4 form a pair of transmission gates while transistors M5 and M6, M7 and M8 create a pair of inverters. The S control signal is applied to the gates of M1 and M4 and the $\overline{S}$ control signal is applied to the gates of M2 and M3. The input signal is applied to the first transmission gate (M1 and M2) while the output is taken from the second inverter (M7 and M8). When $S = 1$ and $\overline{S} = 0$ (input signal is inside the decimation band) the input is decoupled from the first transmission gate and the output Q remains at its previous value. When $S = 0$ and $\overline{S} = 1$ (input signal is outside the decimation band) the first transmission gate is enabled and the input is passed to the output Q. The second transmission gate (M3 and M4) is disabled. Table 10 summarizes the overall behavior of the D flip-flop circuit.

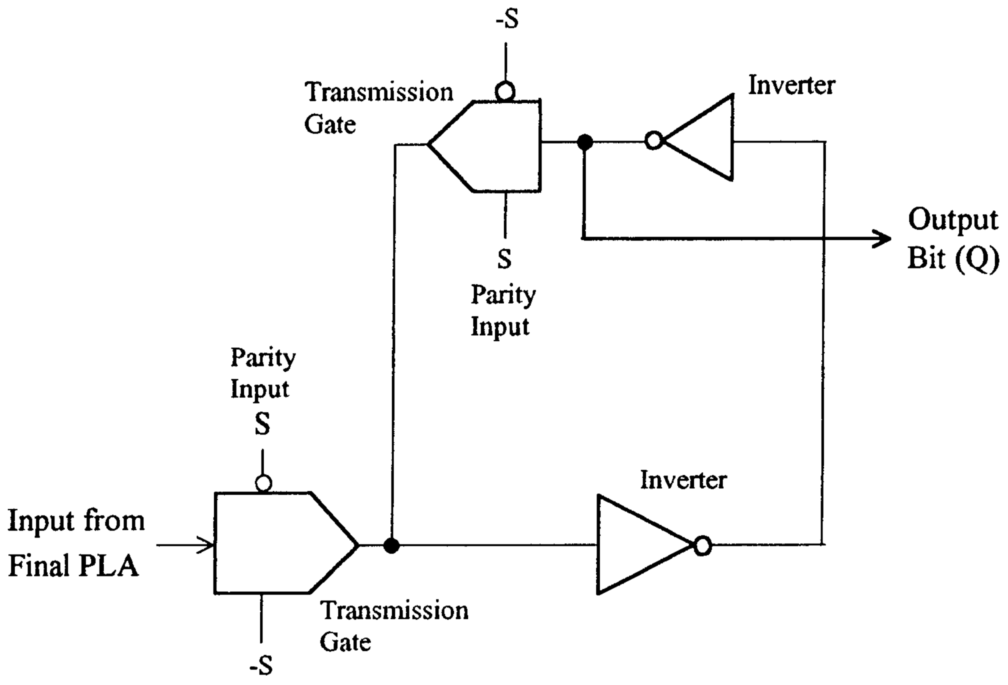


Figure 20. Symbolic Layout of D Flip-Flop Circuit.
(1 per bit)

## E. PLA SUBSYSTEM

Upon successful design of the comparator and latch circuits, the final circuit to design is the PLA subsystem which is composed of a group of modulus PLAs and the final PLA. The modulus PLA transforms the comparator group output (thermometer format) values into a binary format. Table 11, 12 and 13 relates the output of the comparator groups to the output of the modulus PLA for $m_1 = 7$, $m_2 = 8$ and $m_3 = 11$ respectively. The output of the modulus PLAs represent the SNS value of the input signal applied to

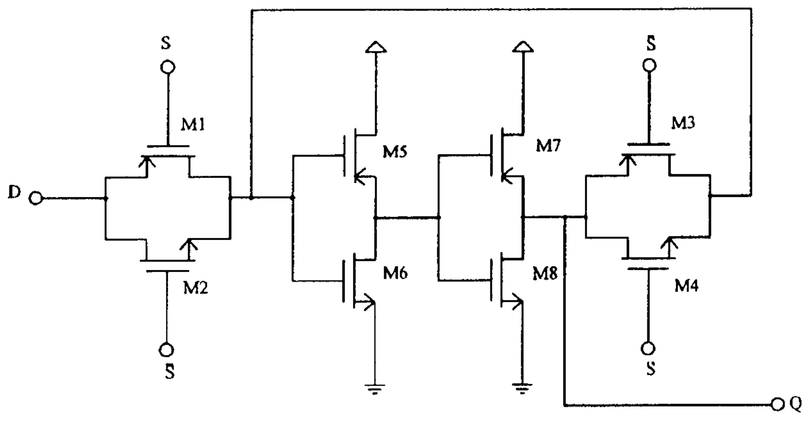


Figure 21. D Flip-Flop CMOS Circuit.

the overall analog-to-digital converter. Using the U.C. Berkeley CAD tool, MPLA, the PLA subsystem can be generated by specifying the inputs and output vectors in a file and then running MPLA. Appendix C is the MPLA input file code for the generation of modulus 7, 8 and 11 PLAs.

The second function of the PLA subsystem is to take the combined modulus PLA outputs for $m_1 = 7$, $m_2 = 8$ and $m_3 = 11$ and translate them into a straight binary output. The SNS binary output of each modulus PLA represents the total number of comparators that are ON at any given time. Table 14 relates the combined modulus PLAs (in decimal code) in SNS format to the straight binary format (in decimal code). Note that this table is only a partial listing of the entire 616 possible combinations. The total number of inputs into the final PLA is ten, thus providing $2^{10} = 1024$ possible output combinations.

| Input Signal D | Parity State S | $\overline{S}$ | Transmission Gate 1 | 2 | D Flip-Flop Output Q |
|---|---|---|---|---|---|
| 0 | 0 | 1 | Enabled | Disabled | 0 |
| 1 | 0 | 1 | Enabled | Disabled | 1 |
| 0 | 1 | 0 | Disabled | Enabled | Q |
| 1 | 1 | 0 | Disabled | Enabled | Q |

Table 10. Overall Behavior of D Flip-Flop Circuit.

| Comparator Group$_7$ Outputs | | | | | | Mod 7 PLA Outputs | | |
|---|---|---|---|---|---|---|---|---|
| $C_6$ | $C_5$ | $C_4$ | $C_3$ | $C_2$ | $C_1$ | $Y_3$ | $Y_2$ | $Y_1$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Table 11. Truth Table for Modulus 7 PLA.

However, the dynamic range for this 9-bit SNS folding ADC is 616, given by equation (2), and therefore the entire dynamic range has been easily covered. Appendix D is the MATLAB m-file code which generates the truth table for the final PLA [Ref. 7]. The truth table is used in the Mpla input file for the generation of the final PLA. Appendix E contains the MPLA input file for the final PLA.

| Comparator Group$_8$ Outputs | | | | | | | Mod 8 PLA Outputs | | |
|---|---|---|---|---|---|---|---|---|---|
| $C_7$ | $C_6$ | $C_5$ | $C_4$ | $C_3$ | $C_2$ | $C_1$ | $Y_3$ | $Y_2$ | $Y_1$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 12. Truth Table for Modulus 8 PLA.

| Comparator Group$_{11}$ Outputs | | | | | | | | | | Mod 11 PLA Outputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_{10}$ | $C_9$ | $C_8$ | $C_7$ | $C_6$ | $C_5$ | $C_4$ | $C_3$ | $C_2$ | $C_1$ | $Y_4$ | $Y_3$ | $Y_2$ | $Y_1$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

Table 13. Truth Table for Modulus 11 PLA.

| MODULUS PLA | | | 9-BIT OUTPUT |
|---|---|---|---|
| MOD 7 | MOD 8 | MOD 11 | FINAL PLA |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 |
| 6 | 7 | 7 | 7 |
| 5 | 7 | 8 | 8 |
| 4 | 6 | 9 | 9 |
| 3 | 5 | 10 | 10 |
| 2 | 4 | 10 | 11 |
| 1 | 3 | 9 | 12 |
| 0 | 2 | 8 | 13 |
| 0 | 1 | 7 | 14 |
| 1 | 0 | 6 | 15 |
| . | . | . | . |
| . | . | . | . |

Table 14. Truth Table for Final PLA (Partial Listing).

# V. VLSI IMPLEMENTATION

Once the various circuits are defined for the three channels $m_1$=7, $m_2$=8 and $m_3$=11 in Chapter IV, the VLSI layout is accomplished using the Magic CAD tool. The design rule checker (DRC) of Magic for the 2μ CMOS N-well technology is used to verify correct layout of the circuits. The discussion of the layout of the circuits follows the same outline in Chapter IV. Each individual circuit is examined and then the three separate moduli channels are shown. Finally, the three channels are brought together with the PLA subsystem, parity circuit and the output latch circuit. Figure 22 illustrates the block diagram of the CMOS implementation. An example nfet CMOS transistor is shown in Figure 23. Basic transistor features, gate length and width, dimension and CMOS layer representation are labeled.

Two metal layers are used in the design. The minimum size width for the power and ground conductors are chosen such that the current density of the particular conductor will not exceed the current limit for either metal layers. A current limit of 0.6 milliamps per micron of width for metal one and 1.15 milliamps per micron of width for metal two is used. Exceeding the current limit can cause electromigration which will eventually cause an opening in the metal conductors. Another consideration in determining the minimum size width for the power and ground conductors is the voltage drop from the power and ground input pads to the circuit. That is, in order for the circuit to operate properly it must receive the correct voltage. However, the voltage drops are calculated and determined to be insignificant due to the very short distant from the input

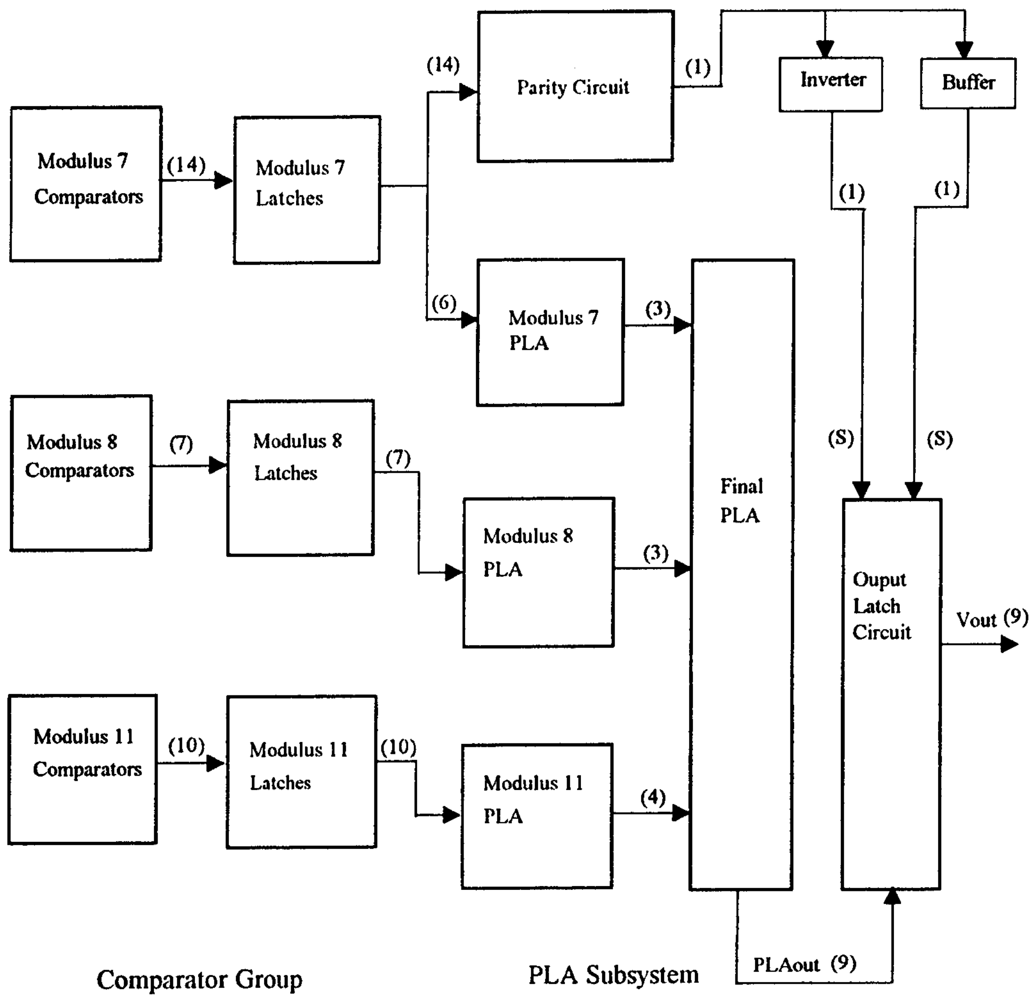
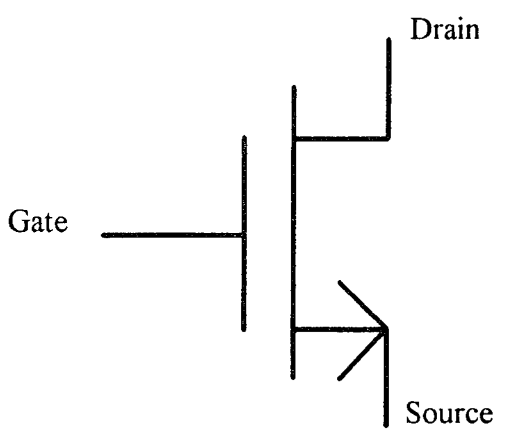


Figure 22. Block Diagram of VLSI Implementation.

pads to the circuit. Substrate contacts are used extensively throughout the layout in order to ensure that the substrate-well junction remained reverse biased, therefore preventing latch-up. Unlike the analog preprocessing circuits, the transistors in the following circuits are small and there are no resistors or capacitors in the circuits.



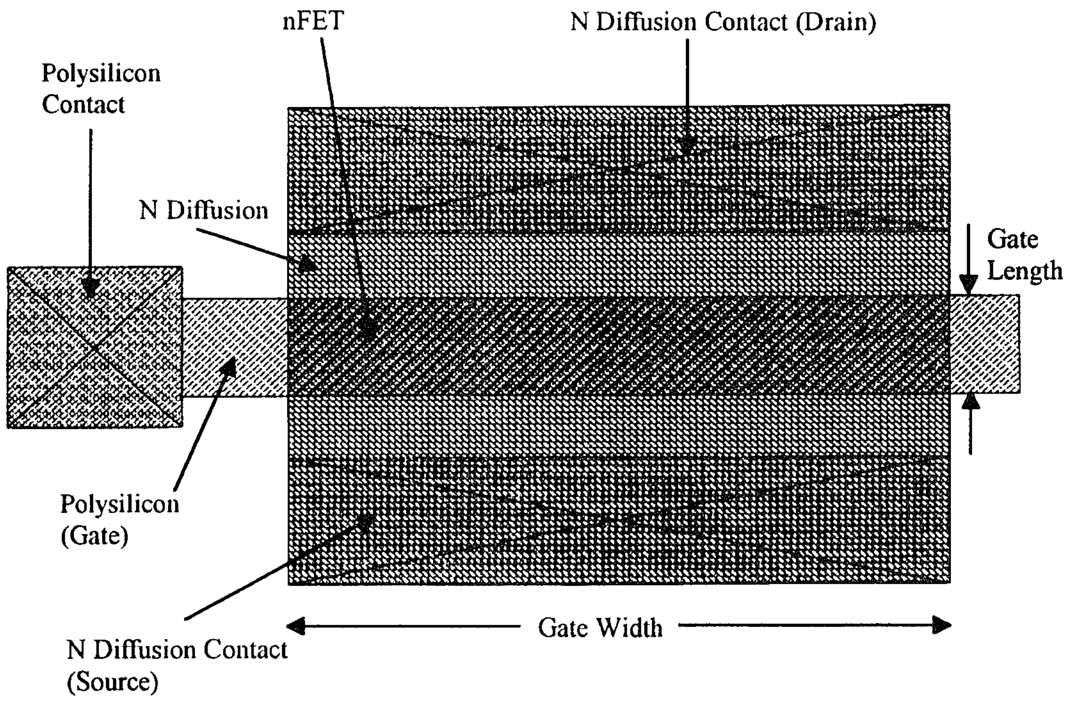


Figure 23. Example CMOS Transistor Layout.

## A. COMPARATOR CIRCUIT

The layout of the comparator circuit consist of eight transistors. The creation of a basic cell incorporating all the transistors in the comparator circuit is accomplished using the Magic CAD tool. Then this basic cell is copied as necessary to obtain the proper number of comparators in each of the three moduli. Once this has been done, $V_{in}$, $V_{bias}$, $V_{dd}$, $V_{ss}$ and GND are interconnected to each comparator circuit. Finally, a separate lead is connected to the gate of M2 of each comparator circuit for the threshold voltage $V_t$ to be applied. Figure 24 shows the CMOS implementation of a single comparator circuit. The labeling of the transistors matches that of Figure 15. Figure 25 represents the CMOS implementation of the modulus 8 comparator circuit. Note the seven single comparator circuits connected in parallel.

## B. LATCH CIRCUIT

The CMOS implementation of the latch circuit described in Chapter IV is shown in Figure 26. Component labeling coincides with that present in Figure 17. The layout consist of ten transistors. The input is connected to the gates of M1 and M4. The control signal $\overline{\text{CLK}}$ is connected to the gates of M2 and M9, where as CLK is connected to the gates of M3 and M8. The output is taken from the drain of M6. Once the basic latch cell has been completed, the cell is copied as necessary to obtain the proper number for each moduli. Then $V_{dd}$, GND, CLK and $\overline{\text{CLK}}$ lines are interconnected to each latch cell. Finally, the comparator circuit described in the previous section is added by using the Magic command **:getcell**. Figure 27 represents the CMOS implementation of modulus 8

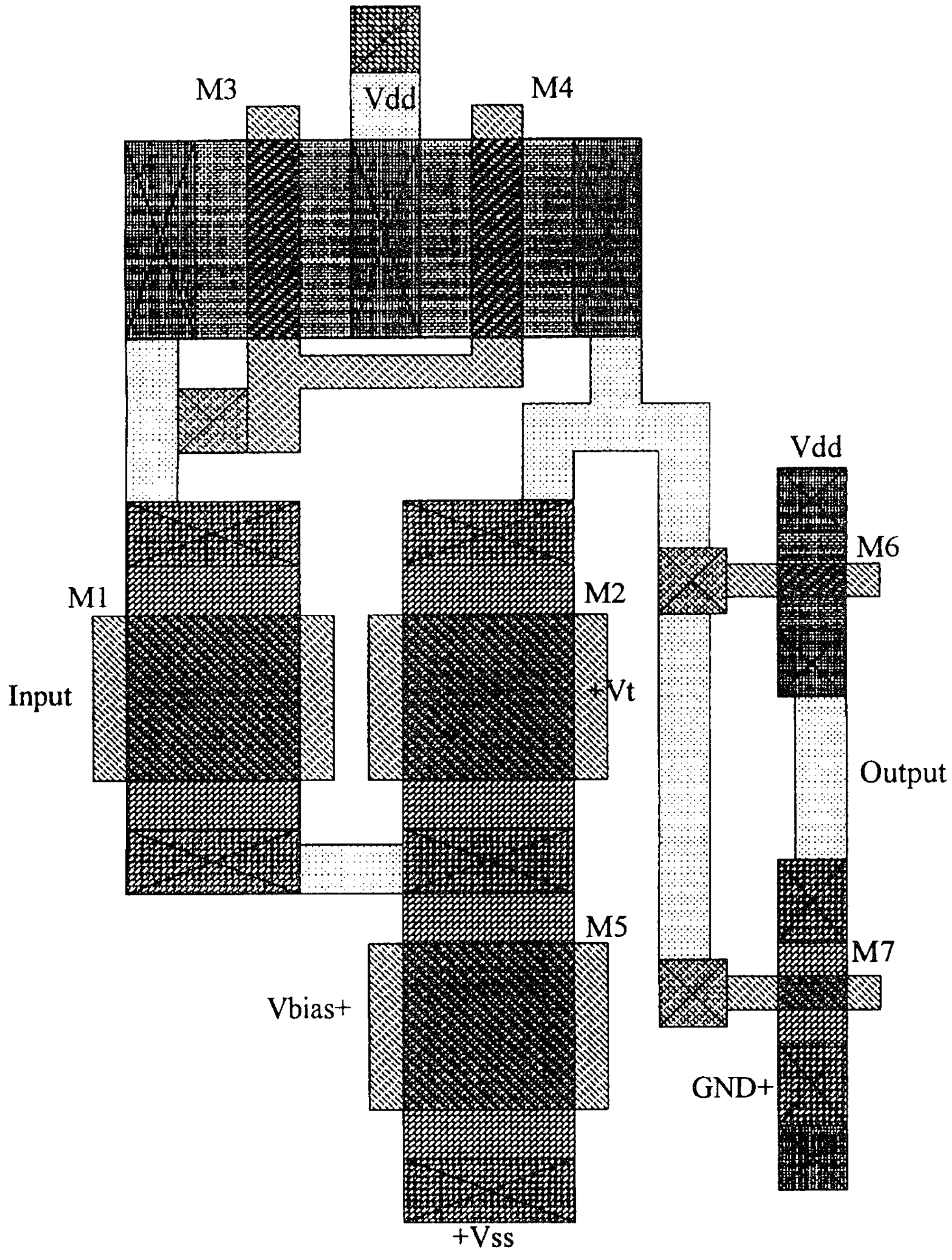


Figure 24. CMOS Implementation of Comparator Circuit.

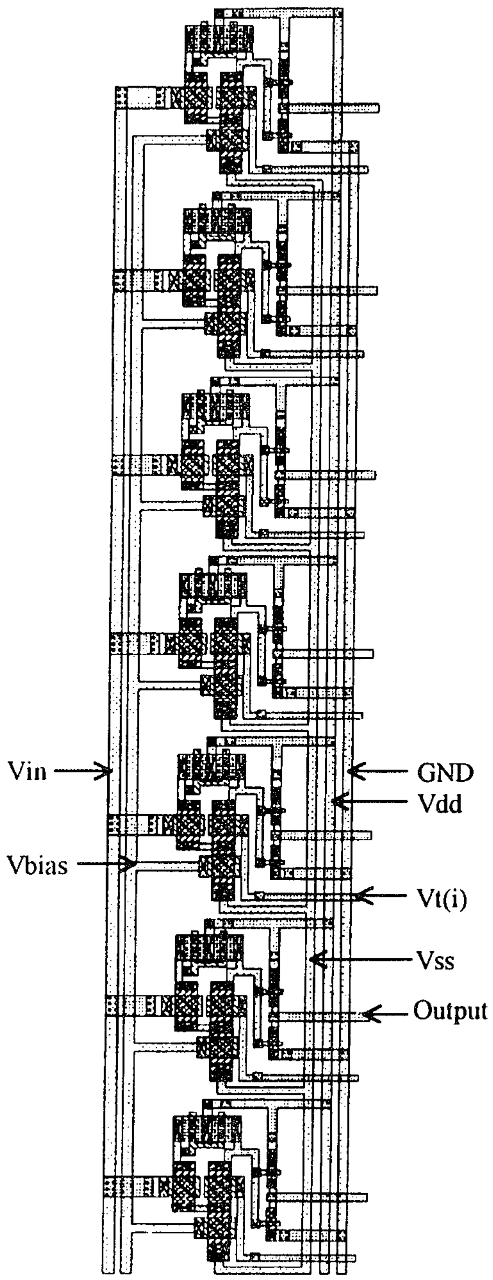


Figure 25. CMOS Implementation of Modulus 8 Comparators.

comparator group. Note how the output of each separate comparator circuit feeds directly into the input of the latch circuit. Also note that there are seven ($m_i$ - 1) comparators and latches.

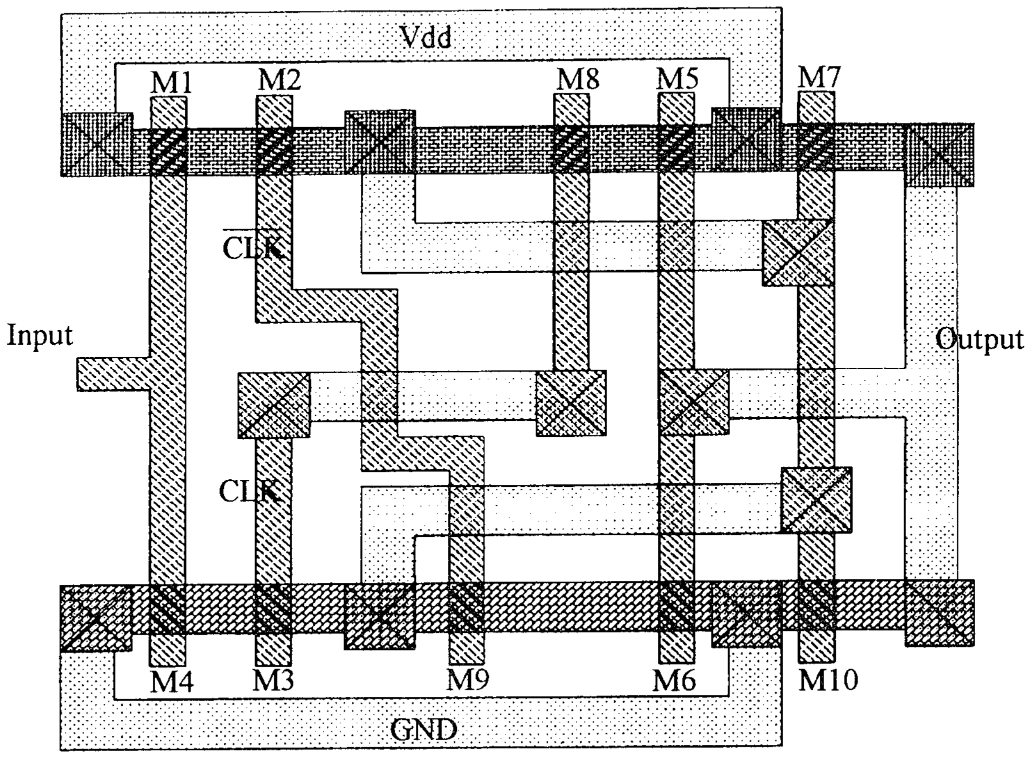


Figure 26. CMOS Implementation of Latch Circuit.

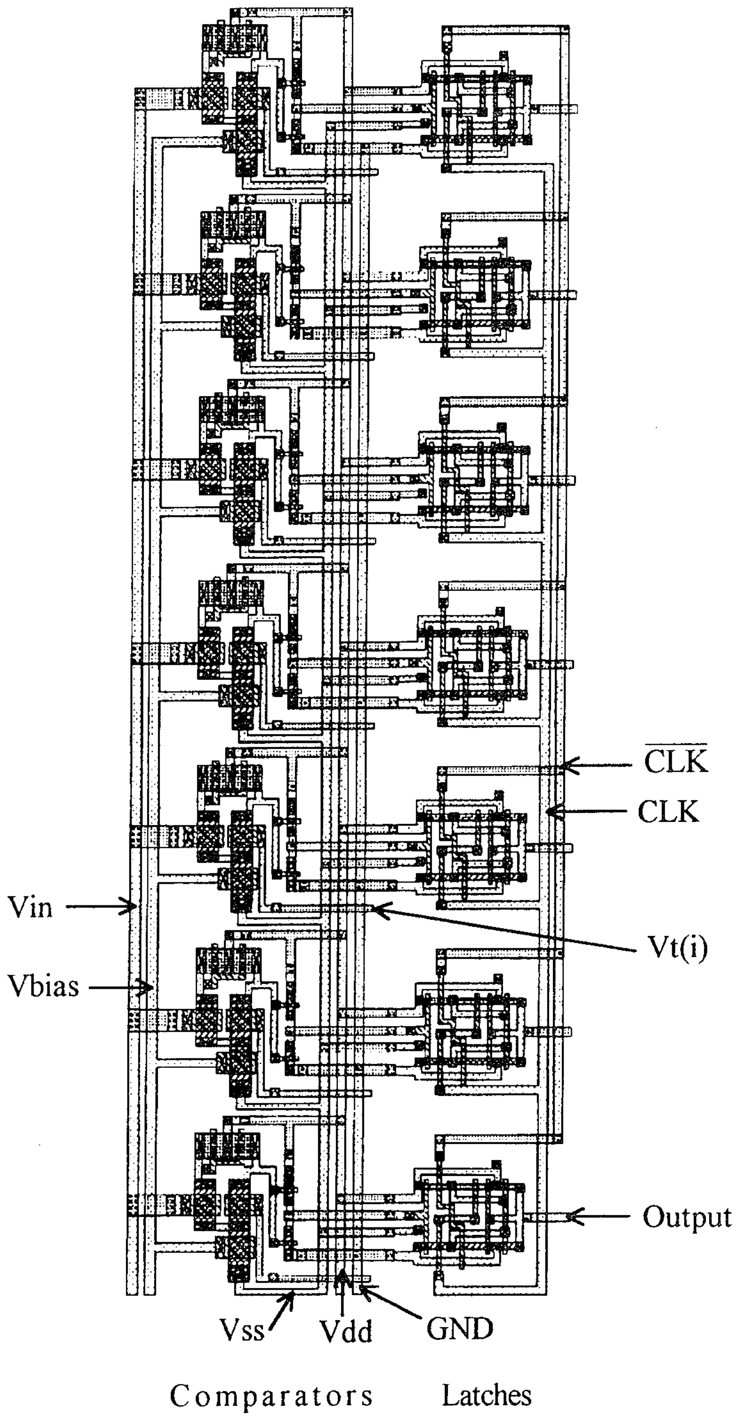


Figure 27. CMOS Implementation of Comparator Group 8.

## C. PARITY CIRCUIT

The layout of the parity circuit consist of 13 2-input XOR gates and one inverter. Each XOR gate contains six transistors and the inverter has two transistors. Figure 28 is the CMOS implementation of the 2-input XOR gate and inverter. The labeling matches that present in Figure 19 for the XOR gate. The first input is connected to the gates of M1, M2, M5 and the source of M3. The second input is connected to the sources of M5 and M6 and the gates of M3 and M4. Source voltage ($V_{dd}$) is connected to the source of M1 and ground (GND) is connected to the source of M2. The output is taken from the drains of M5 and M6. Once the basic XOR gate cell has been created, it is copied 13 times and layed out in a tree-like structure. Finally, the output of the last XOR gate is passed through an inverter. Figure 29 shows the completed layout of the parity circuit which resembles the symbolic layout of Figure 23. Note that all $V_{dd}$ and GND lines are interconnected.

## D. OUTPUT LATCH CIRCUIT

Implementing the sample and hold circuit is accomplished in a similar manner as the latch circuit. Figure 30 represents the CMOS implementation of the D flip-flop discussed in Chapter IV. Again, the labeling coincides with that of Figure 26. The layout consist of eight transistors, four configured as a pair of inverters and four configured as a pair of transmission gates. The input is applied to the source of M1 and M2 and the output is taken from the source of M3 and M4. The control signal S is applied to the gates of M1 and M4, and $\overline{S}$ is applied to the gates of M2 and M3. Source voltage $V_{dd}$ is

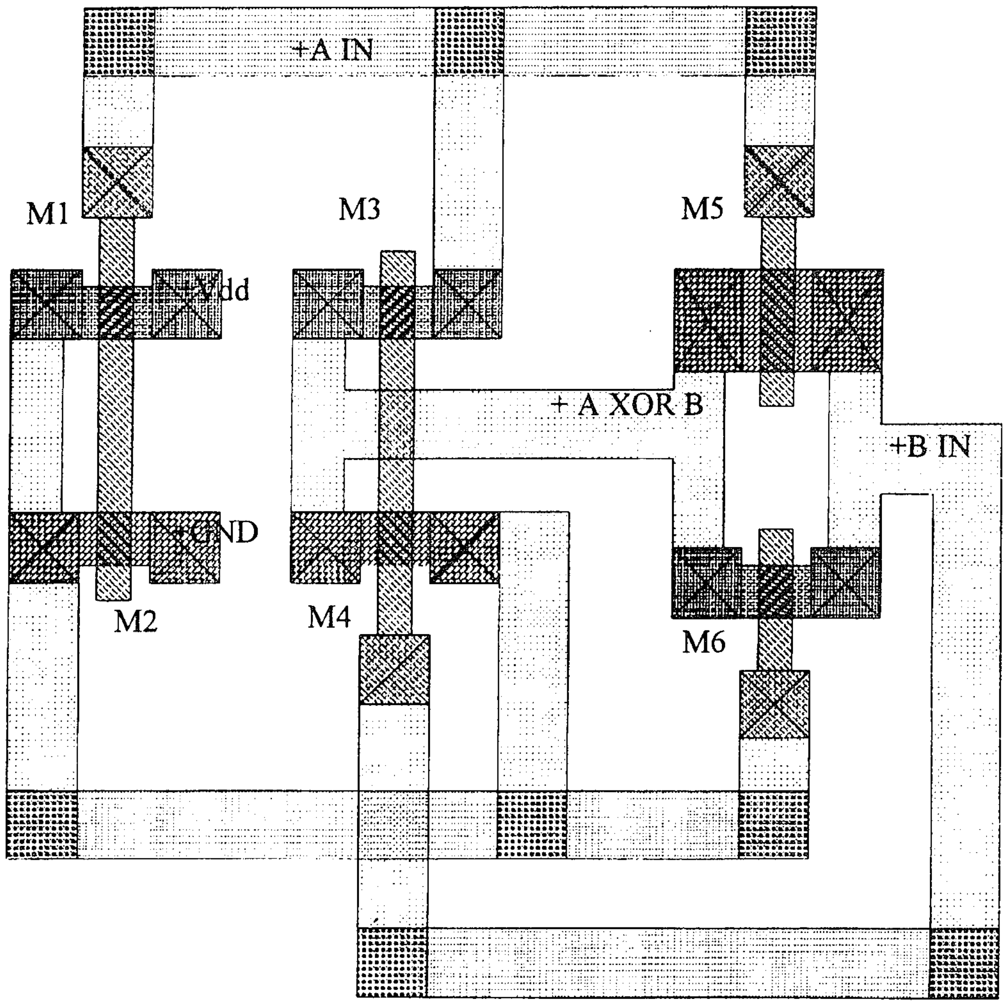


Figure 28. CMOS Implementation of 2-Input XOR Gate.

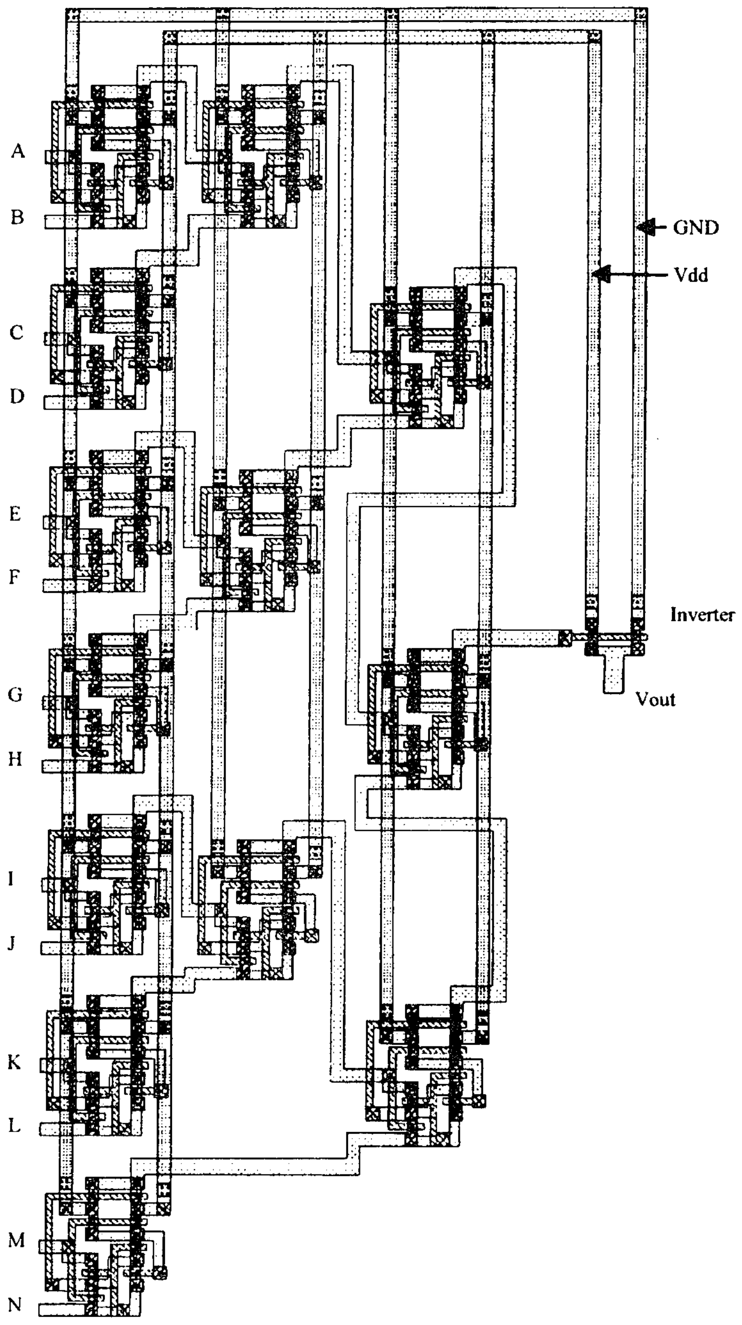


Figure 29. CMOS Implementation of Even-Parity Circuit.

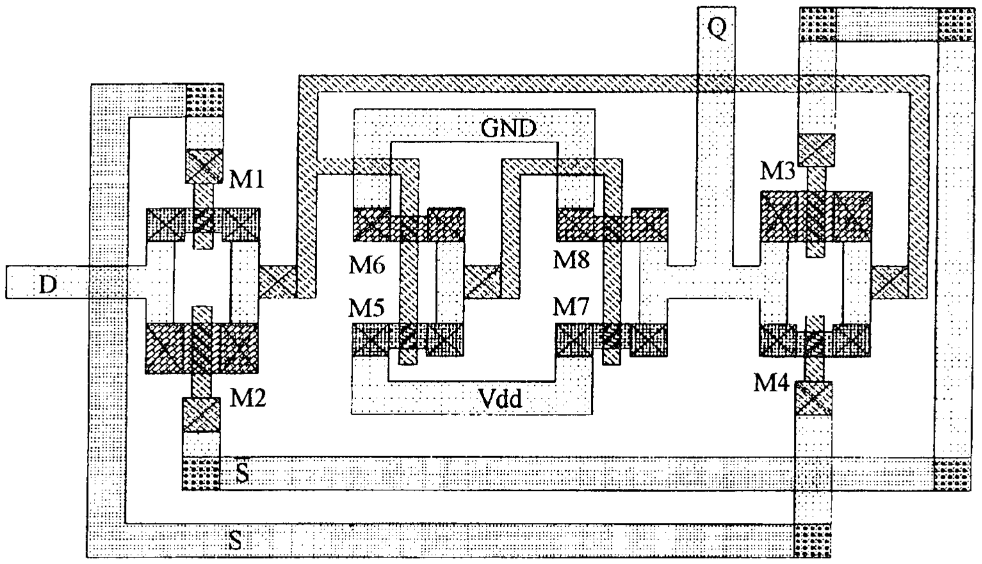


Figure 30. CMOS Implementation of D Flip-Flop Circuit.

connected to the source of M5 and M7 and ground GND is connected to the drains of M6 and M8.

## E. PLA SUBSYSTEM

The layout of the PLA subsystem is straight forward using the U.C. Berkeley CAD tool, MPLA. Appendix C part A, B and C is the MPLA input file for the generation of modulus 7, 8 and 11 PLAs respectively. The format of the MPLA file consist of comments (lines beginning with '#' ) and controlling information (lines beginning with '.' ). The controlling information given is the inputs (**.i** <number of inputs>), outputs (**.o** <number of outputs>), and product terms (**.p** <number of product terms>). Then, what follows is the truth table of the desired PLA. The end of the MPLA file is indicated with **.e**. The parameters **.ilb** indicate the input labels will be on the left-bottom AND plane and **.orb** indicate the output labels will be on the right-bottom OR plane. Figure 31, 32 and 33 is the CMOS implementation of the modulus 7, 8 and 11 PLAs respectively. The inputs are labeled $X_i$ ..$X_1$ and the outputs are labeled $Y_i$ ... $Y_1$. Note the inputs are applied to the AND logic plane on the left side and the outputs are taken from the OR logic plane on the right side of the PLA. Also note the increasing size of the PLAs from modulus 7 to modulus 11. That is, the larger the truth table (Tables 11, 12, 13) the larger is the PLA generated. Figure 34 is the CMOS implementation of the final PLA Even though it is hard to denote, the inputs ($X_{10}$ ... $X_1$) are applied to the AND logic plane on the left side and the outputs ($Y_9$ ... $Y_1$) are taken from the OR logic plane on the right side of the PLA. Note the tremendous size difference between the final PLA and modulus 11 PLA. That is,

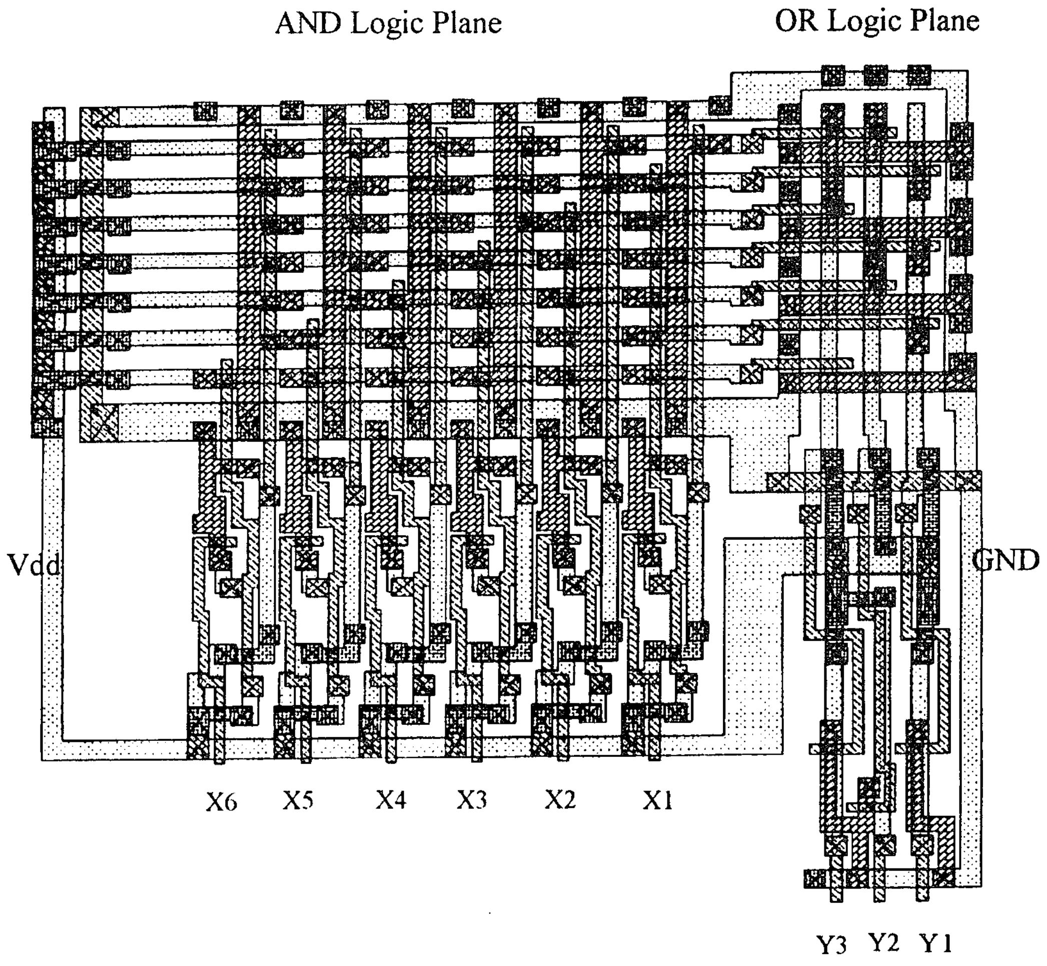


Figure 31. CMOS Implementation of Modulus 7 PLA.

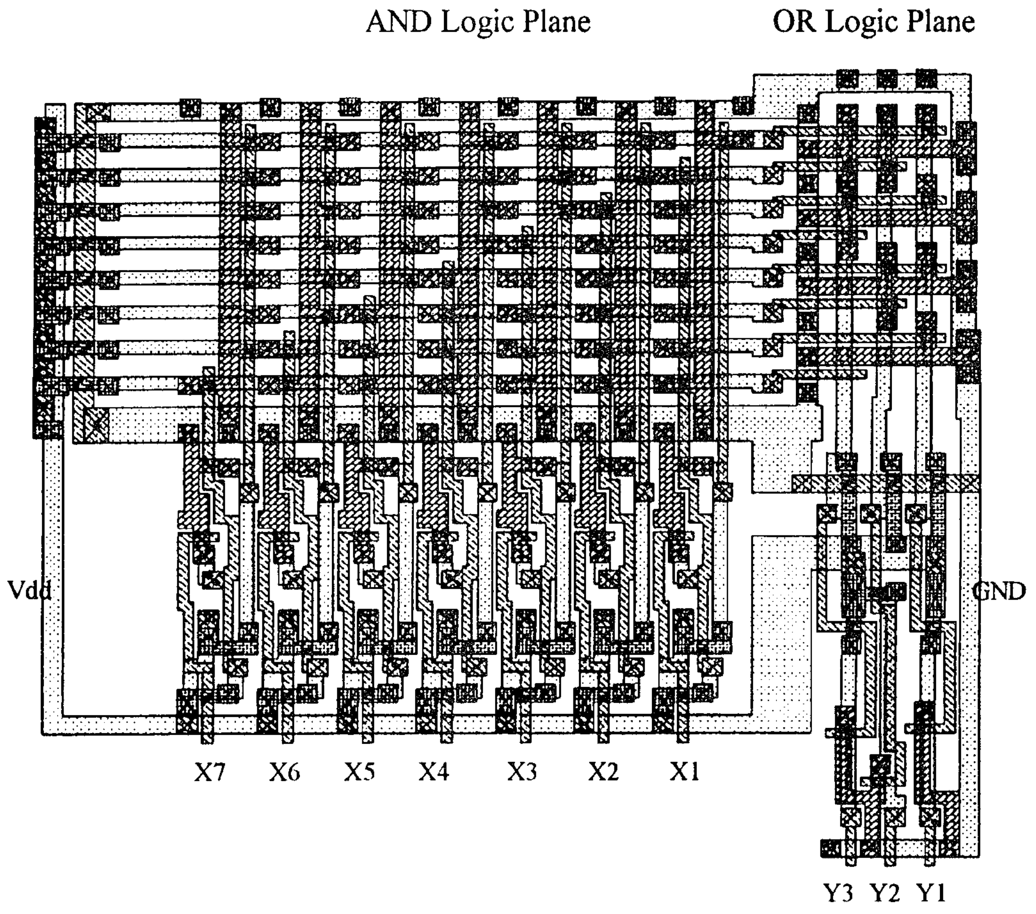


Figure 32. CMOS Implementation of Modulus 8 PLA.

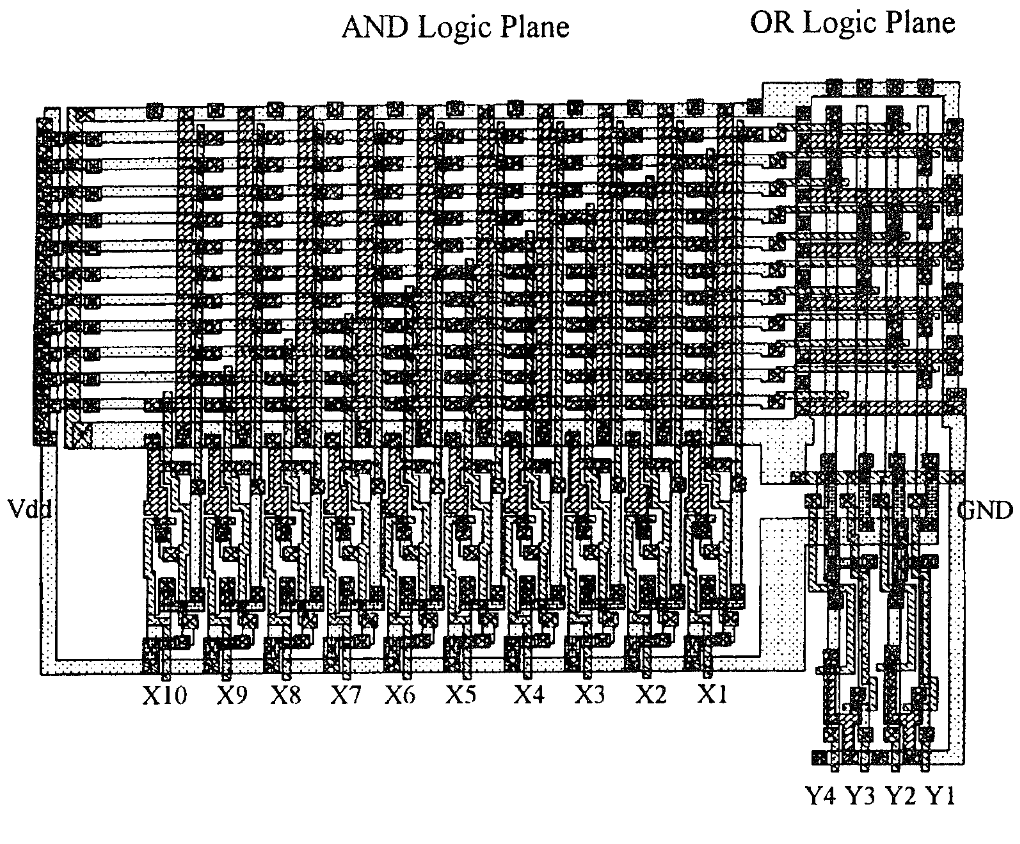


Figure 33. CMOS Implementation of Modulus 11 PLA.

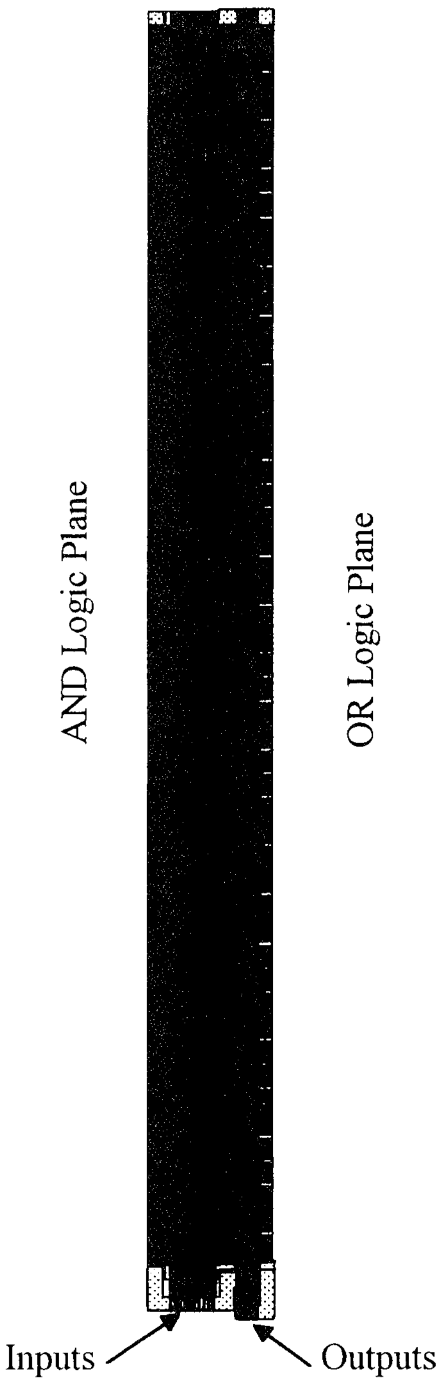


Figure 34. CMOS Implementation of Final PLA.

the truth table of the final PLA consist of 616 entries whereas modulus 11 has only 11 entries. Appendix D is the MATLAB m-file code to generate the truth table for the final PLA [Ref. 7]. Appendix E is the MPLA input file for the final PLA [Ref. 7].

## F. SYSTEM DESIGN

Implementing the circuits discussed in the previous sections into the system design of the ADC is accomplished in a hierarchical layered manner. First, the comparator group is loaded in a Magic cell. Then the associated modulus PLAs, parity circuit, output latch circuit and the final PLA are added by using the Magic command **:getcell**. Figure 35 is the floor plan for the layout of all three channels and the final PLA. The major goal in the layout is the minimization of the chip area. Figure 36 is the CMOS implementation of the floor plan. Note the similarities to the block diagram of Figure 27. A common source voltage $V_{dd}$ and $V_{ss}$ and ground GND interconnect are used with taps taken off to each of the circuits. Also, a common control line for CLK and $\overline{\text{CLK}}$ are used with taps taken off where necessary. Note the output the parity circuit is jointly passed through a buffer and an inverter to ensure proper tir g and phase relationship for the control signals, S and $\overline{S}$ used in the output latch circu The overall size of the CMOS layout is 6.513mm by 1.254mm.

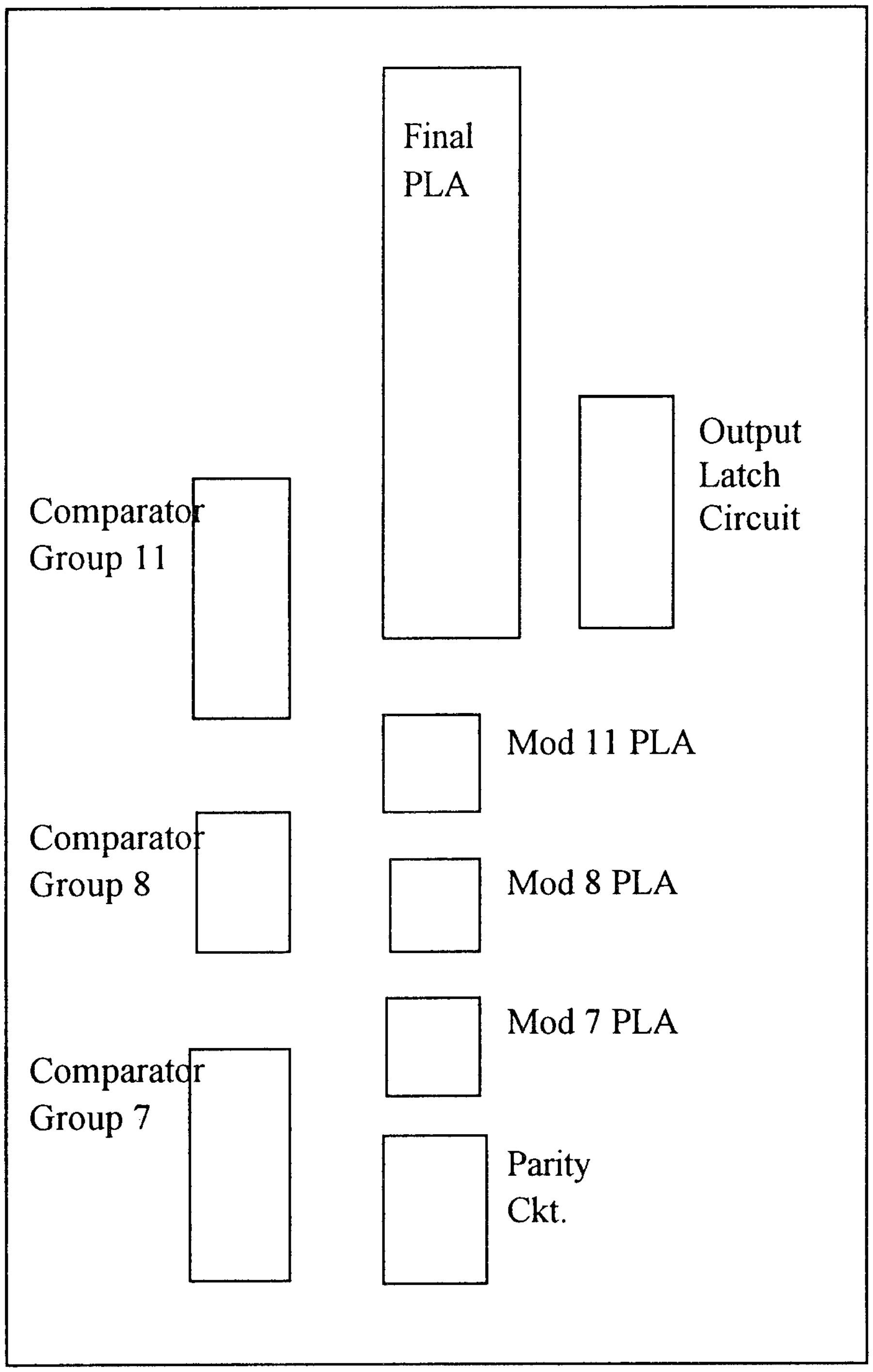


Figure 35. Floor Plan of VLSI Architecture.

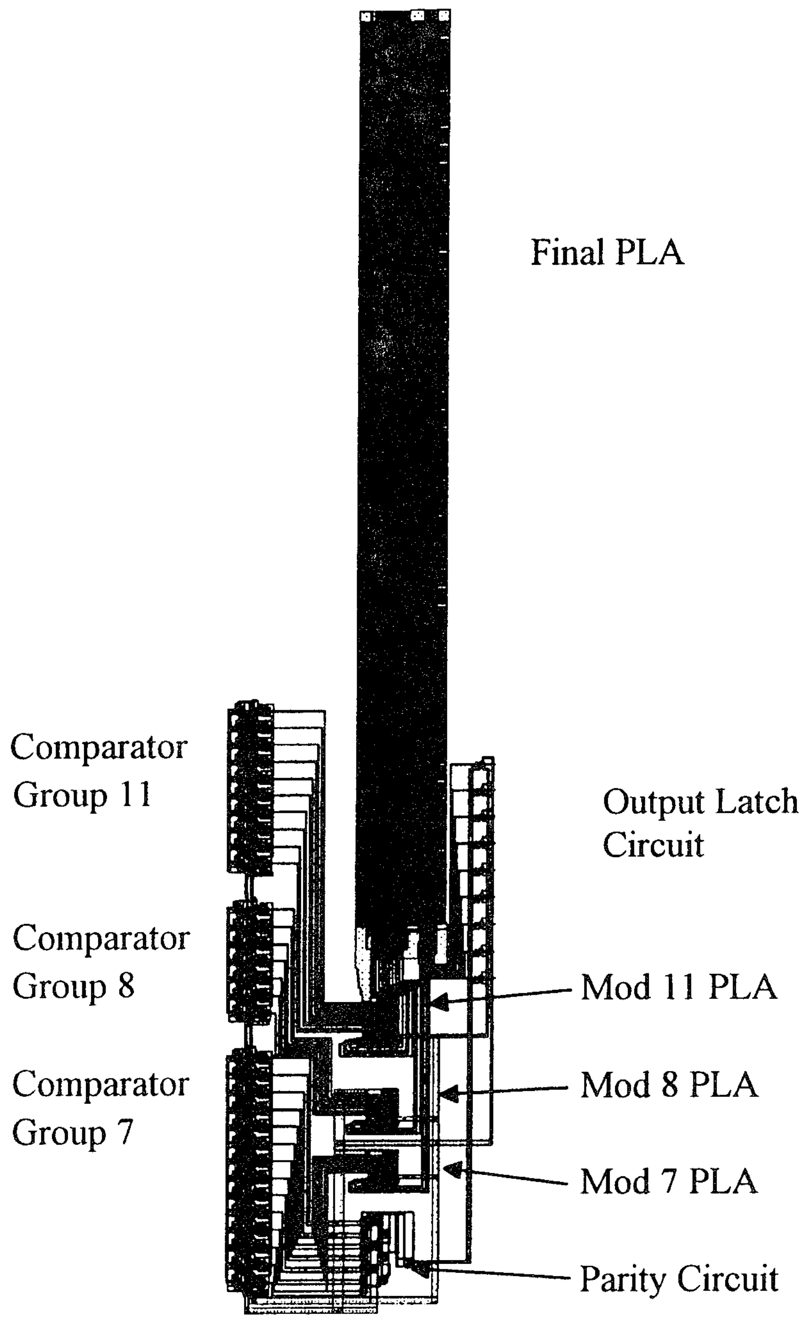


Figure 36. CMOS Implementation of Floor Plan.

# VI. SIMULATIONS

Simulations are conducted of each circuit in order to verify proper implementation of the circuit in CMOS, as well as to compare the CMOS results to the previous BJT architecture research. After the CMOS VLSI layout is complete for each circuit, it is extracted and simulated prior to proceeding to the next level of integration. Finally, simulations are completed on the entire SNS 9-bit folding ADC. Both DC and transient analysis of the circuits are investigated along with power estimation.

## A. CIRCUIT SIMULATION METHODOLOGY

A repetitive method is used for simulating the layout of each circuit design. A simulation flow chart is shown in Figure 37. Once the layout is complete for each circuit, the inputs, outputs, power, ground and any control signals must be labeled. The extraction style for Magic is verified and set to 1.0 (lambda=1.0). This results in a *unit* in Magic corresponding to 1 micron. This is important for the extraction process which bases the size on the layout area in Magic. That is, if a transistor is layed out in Magic with gate size of 4μ by 2μ then the extraction of that transistor must be the same. Once the extraction style is set, the layout is extracted using the Magic command **:extract**. This produces a **.ext** file corresponding to the layout.

Upon successful extraction of the layout, either the program *ext2spice* or *ext2sim* is invoked. The program *ext2spice* is used to convert the **.ext** file to the **.spice** file used by HSPICE. The new **.spice** file contains transistor instantiations and any internodal coupling and substrate capacitance. Appendix F part A is an example **.spice** file of the

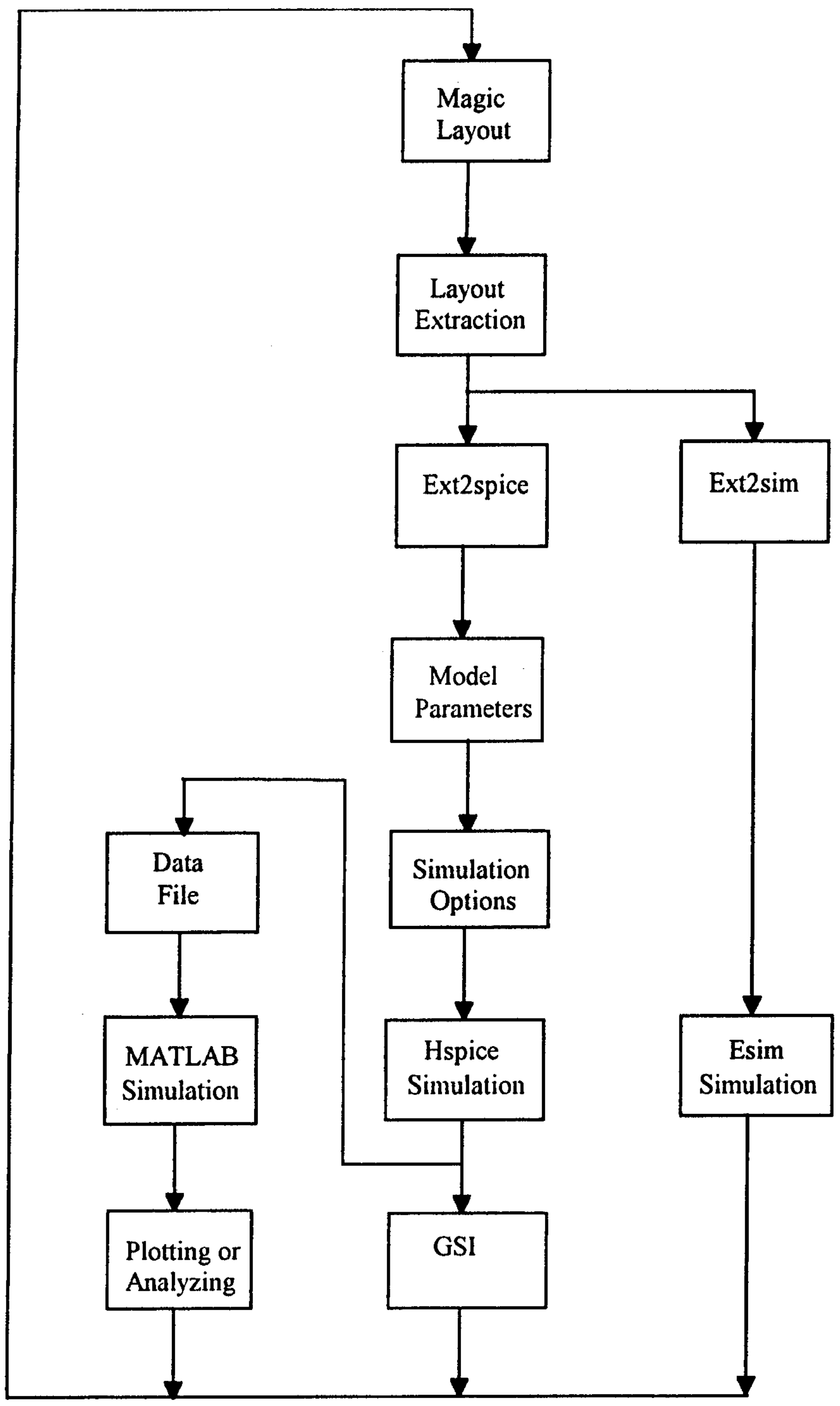


Figure 37. Simulation Flow Chart.

comparator group for modulus 8. Note that the file is not ready for simulation. The model parameters for the transistors must be added along with the source voltages $V_{dd}$ and $V_{ss}$, control signals CLK and $\overline{\text{CLK}}$, threshold voltages $V_t$ and simulation options.

Appendix F part B is the modified **.spice** file ready for simulation using HSPICE. Note the *.probe* statement at the end of the file which stores the simulation results in a graph data file (binary format) to be used by the Graphical Simulator (GSI). Also note the *.print* statement which prints numeric analysis results (ASCII format) to the screen, or the results can be directed to a data file. HSPICE is invoked from the unix command prompt with one of the following commands:

% hspice demo.sp *-m* 3000000 **or**

% hspice demo.sp *-m* 3000000 > tmpdata *&*.

The first method, the user invokes HSPICE with an input file demo.sp. The size of the user work area (swap space) in memory (3000000) is set by using *-m* switch. This is necessary when the input file contains a large number of transistors and HSPICE requires more working memory than the default setting (250000). One graph data file is created for each type of analysis specified in the input netlist file. For example, **.tr#** for a transient analysis, **.sw#** for a DC sweep, and **.ac#** for an AC sweep. Using GSI the simulation results can be verified for proper functional operation. [Ref. 12]

The second method, the input file (demo.sp) is invoked with increased memory and the output results (ASCII format) is directed to the file *tmpdata*. The job is executed in the background (*&*) so the screen and keyboard can still be used. This method is best used when the *.print* statement is used in the input netlist file. The numeric analysis results can then be modified into a MATLAB m-file and the results analyzed using MATLAB. The status of the simulation run can be verified by examining the **.st0** file. If the results of the simulation are not satisfactory, the layout is modified, extracted and simulated using the desired method above until a satisfactory run is obtained.

The program *ext2sim* is used to convert the **.ext** file into a **.sim** file. Once the file has been converted to a **.sim** file, *esim* is invoked from the unix command prompt with the following call: % *esim <filename.sim>*. Again the results are analyzed and if they are not satisfactory the method is repeated.

## B. COMPARATOR GROUP DC ANALYSIS

Simulation of the comparator group is accomplished using the method described in the previous section. The comparator group is extracted and an executable spice file is created. A DC analysis of the circuit is performed by applying a piecewise linear voltage $V_{in}$ to the input of the comparator groups. The threshold voltage $V_t$ for each comparator circuit are taken from Table 6. The control signals, CLK and $\overline{\text{CLK}}$ sample at a 10KHz rate. The steady state ideal folding waveform and comparator group outputs from each channel are shown in Figure 38, 39 and 40, and 41 and 42 for $m_1$=7, $m_2$=8 and $m_3$=11 respectively. This yields simulation results that exercises all of the comparator groups, yet falls short of the entire dynamic range of the SNS 9-bit folding ADC. However, the time required to cover the entire dynamic range would be in excess of a week for each moduli. Also, the entire dynamic range of the folding circuit for each moduli has been exercised in

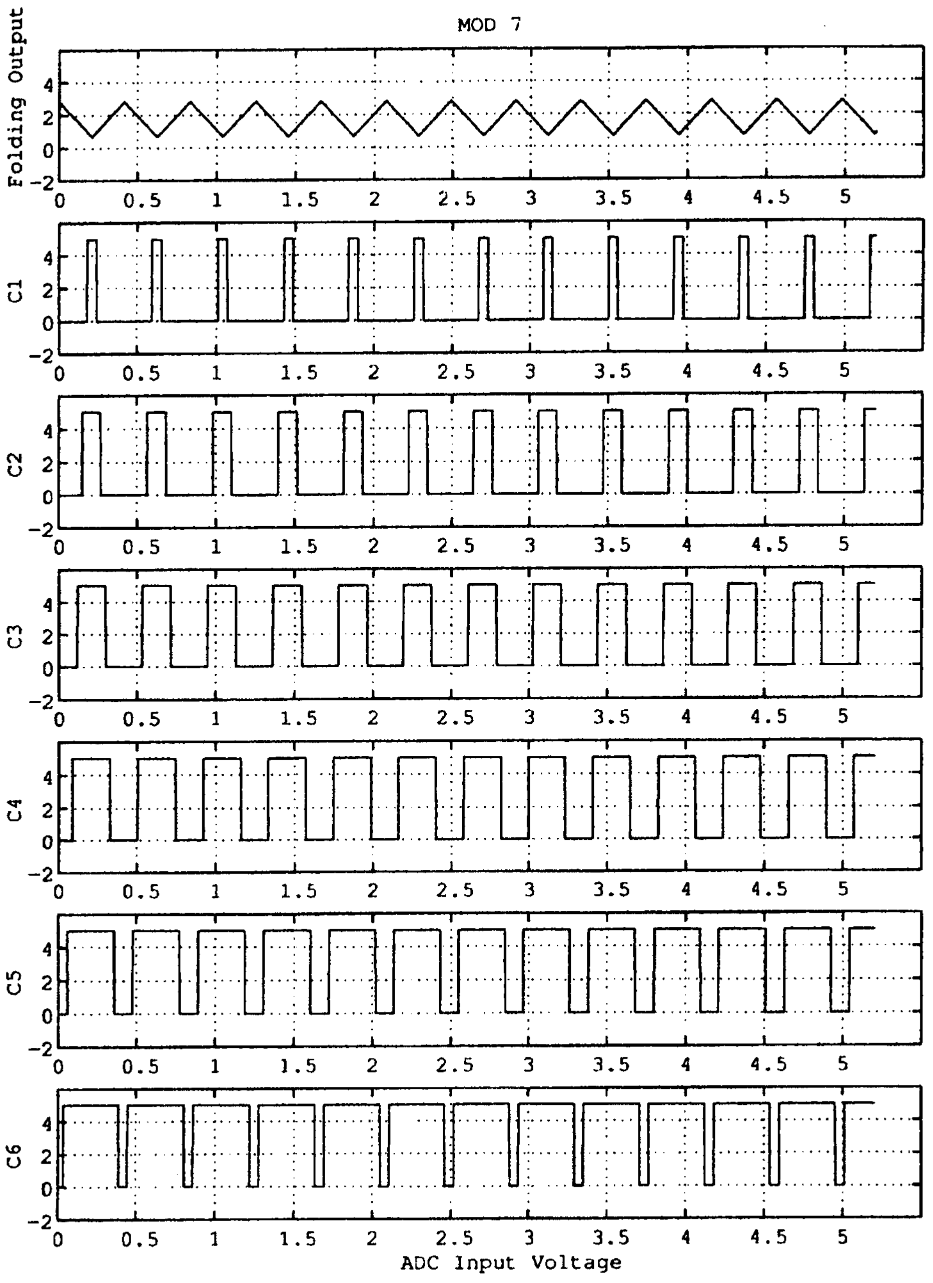


Figure 38. Folding Waveform and Comparator Outputs for $m_1 = 7$.

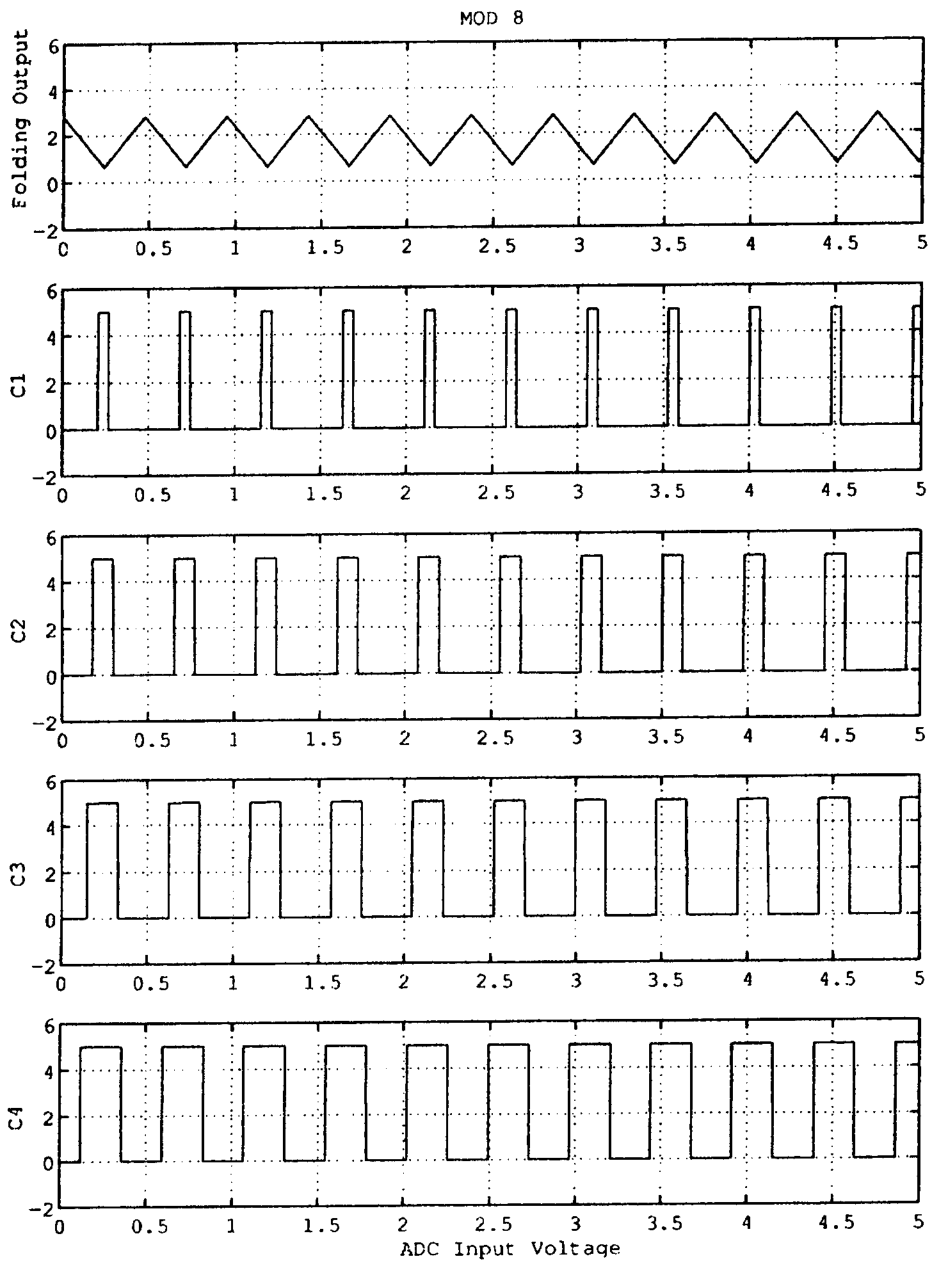


Figure 39. Folding Waveform and Comparator Outputs for $m_2 = 8$.

(Comparators 1 - 4)

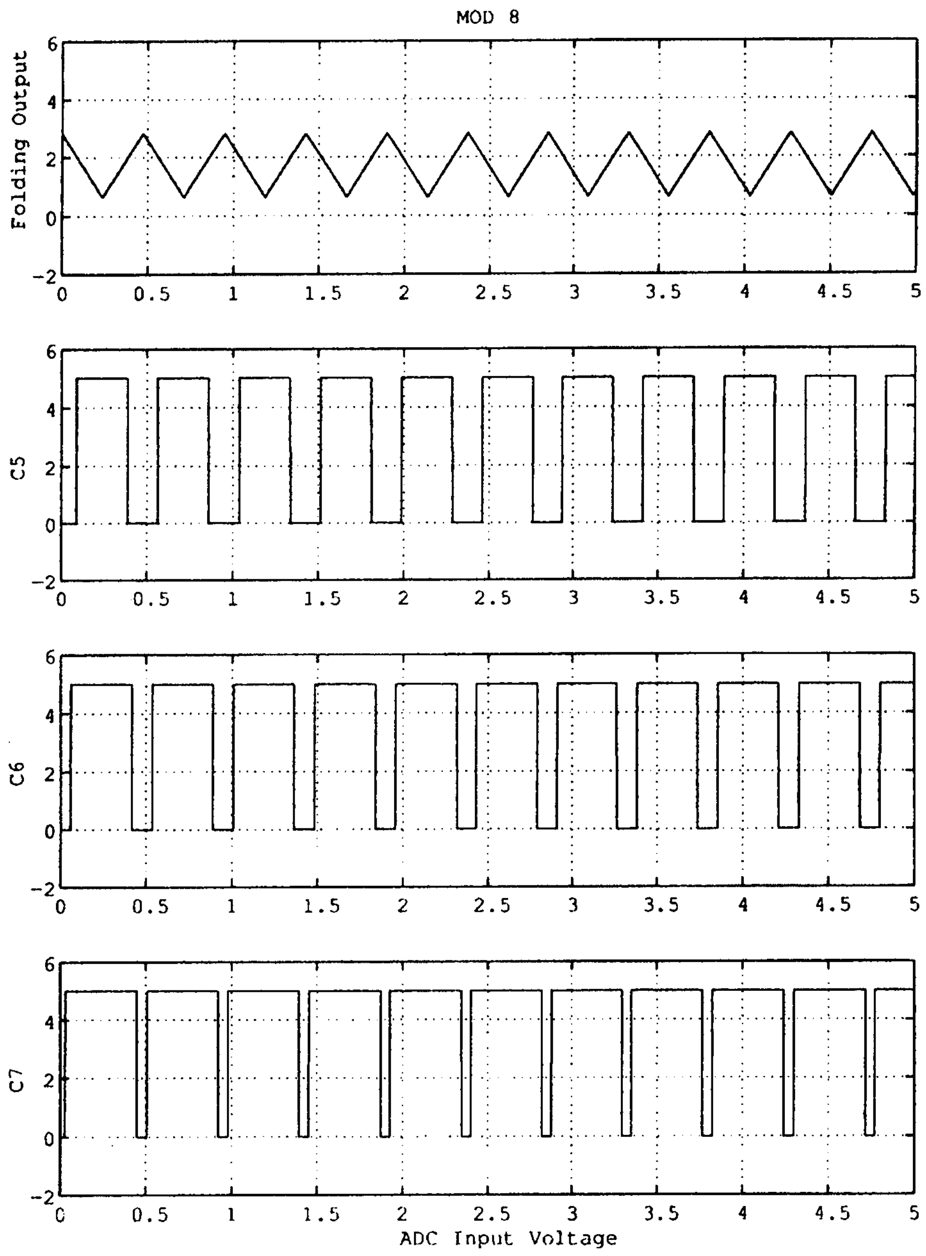


Figure 40. Folding Waveform and Comparator Outputs for $m_2 = 8$.

(Comparators 5 - 7)

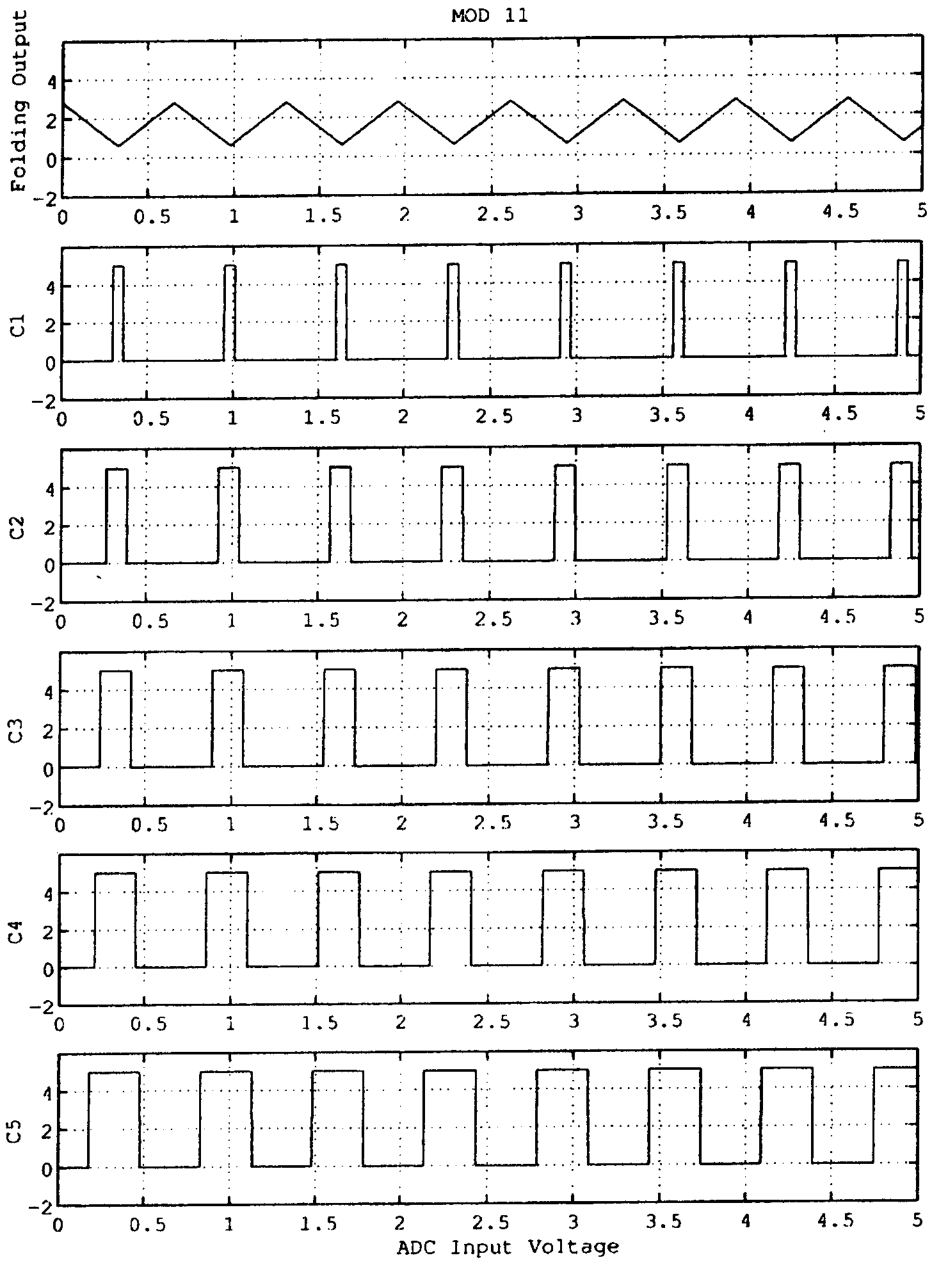


Figure 41. Folding Waveform and Comparator Outputs for $m_3 = 11$.

(Comparators 1 - 5)

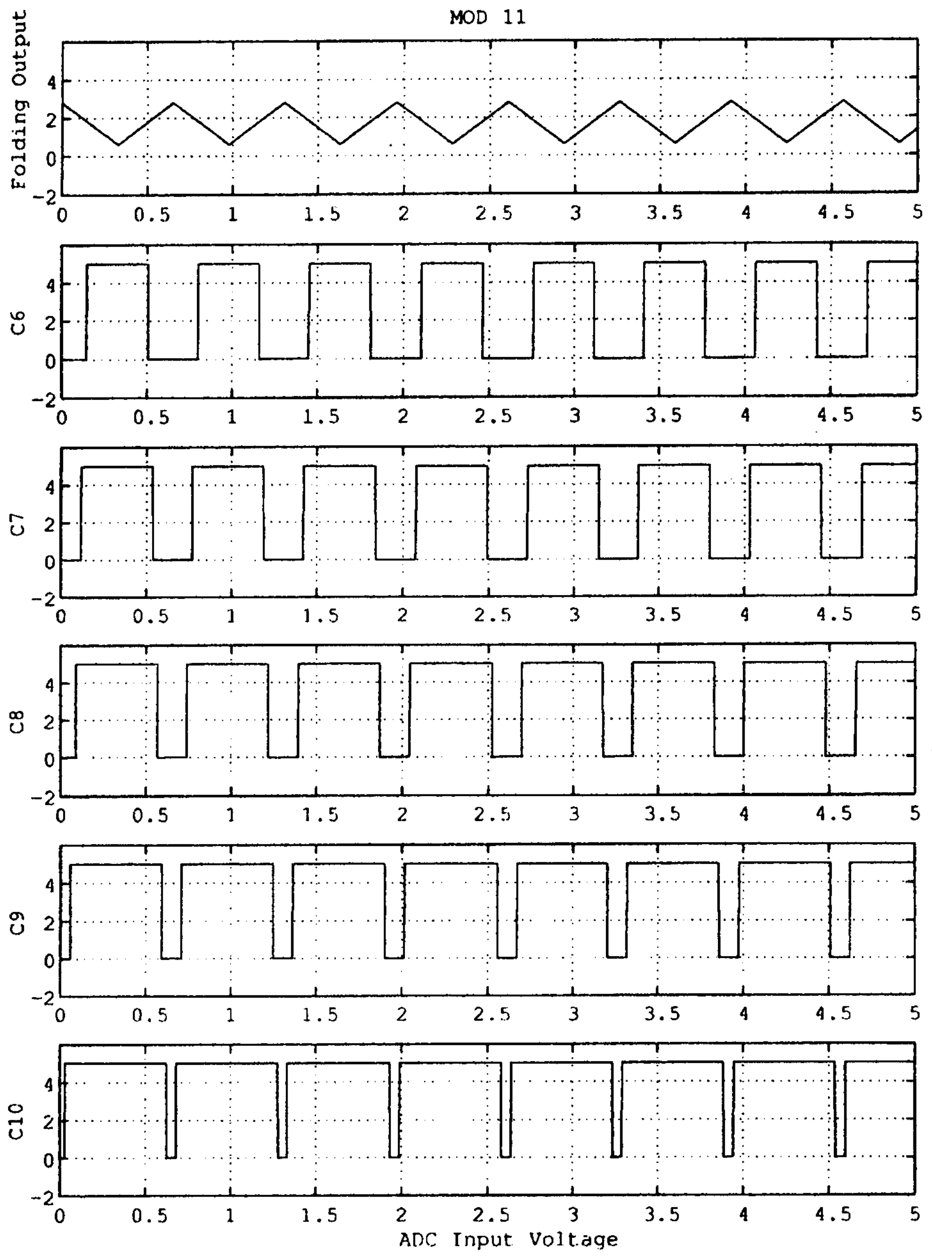


Figure 42. Folding Waveform and Comparator Outputs for $m_3 = 11$.

(Comparators 6 - 10)

previous research [Ref. 15]. The comparator group for each channel generated the proper waveform. Part A of Appendix G is the MATLAB m-file that contains the data derived from the simulation of channel 11. Note, however, only a small portion of the data is present due to the very large size of the data file. Part B of Appendix G is the MATLAB code written to plot the folding waveform and each of comparator group 11 output waveforms.

## C. PARITY CIRCUIT ANALYSIS

Simulation of the parity circuit is accomplished in two phases. First, the 2-input XOR gate and inverter CMOS implementation shown in Figure 28 is extracted and the **.ext** file is converted to a **.sim** file. Then, *esim* is invoked and a simple simulation is conducted to ensure the proper operation of the 2-input XOR gate and inverter prior to connecting all 13 XOR gates in a tree-like structure to create the *even-parity circuit*. Part A of Appendix H contains the *esim* test code and results of the 2-input XOR gate. Note the results match that of Table 8. Part B of Appendix H contains the *esim* test code and results for a basic inverter cell.

Upon satisfactory results of a single 2-input XOR gate and inverter, the entire *even-parity circuit* shown in Figure 29 is extracted and the **.ext** file is converted to a **.sim** file. Part C of Appendix H contains the *esim* test code and results for the *even-parity circuit*. Again note the results match the truth table shown in Table 9. Also note that only the thermometer format output of modulus 7 comparators are exercised as the input to the parity circuit.

## D. OUTPUT LATCH CIRCUIT TRANSIENT ANALYSIS

After successful extraction of the D flip-flop shown in Figure 30, an executable spice file is created. The input signal is selected to be a square waveform ranging from 0.0 to 5.0 volts representing the output of the final PLA. The source voltage $V_{dd}$ is 5.0 volts. The control signals, S and $\overline{S}$, are square waveforms ranging from 0.0 to 5.0 volts and 180° degrees out of phase. The frequency of the control signal is set by the parity circuit. Figure 43 shows the output (Q) of the sample and hold circuit with input signals (D) and control signals (S and $\overline{S}$). Note when the control signal (S) is low, the output (Q) is a mirror image of the input (D). Also when control signal (S) is high, the output (Q) is latched at the last value of the output (Q). Comparing the results shown in Figure 43 to Table 10 indicates that the circuit operates correctly.

## E. PLA SUBSYSTEM ANALYSIS

Finally, the last circuit to be simulated individually is the PLA subsystem. Simulation of the PLA subsystem is straight forward by using *esim*. After the extraction of the PLAs shown in Figure 31, 32, 33 and 34, the **.ext** files are converted to **.sim** files. Then esim is invoked and a simulation is conducted on each separate PLA. The *esim* test code and results are documented in Appendix I part A, B and C for modulus 7, 8 and 11 PLAs respectively. Appendix J contains the *esim* test code and results for the final PLA. Note the results match the truth tables shown in Chapter IV for each separate PLA.

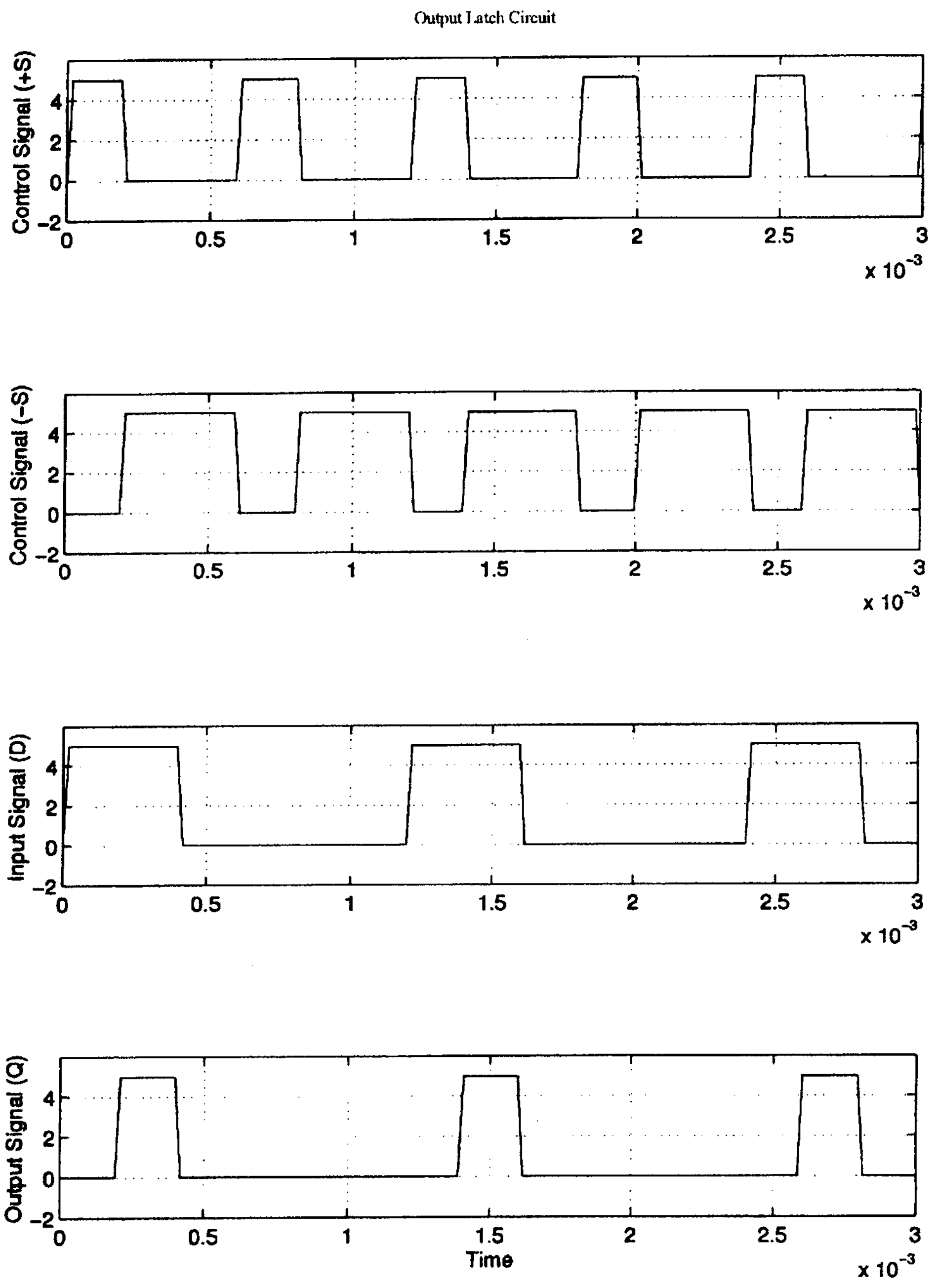


Figure 43. Output of Latch Circuit.

## F. ANALYSIS OF DECIMATION OF ENCODING ERRORS

Analysis of the decimation of encoding errors is accomplished using $2m_i$ comparators for the channel with the *smallest* modulus. For the 9-bit SNS ADC, $2m_1=14$ comparators are required. The comparators are positioned to place a small decimation band around each transition point where the folding waveforms cross the threshold levels. The comparator group for modulus 7 is extracted and an executable spice file is created. A transient analysis of the circuit is performed by applying a piecewise linear waveform $V_{in}$ to the input of the comparator group. This ideal folding waveform represents the folded waveform output of the modulus folding circuits described in [Ref.15]. The threshold voltage $V_t(i)$ for each comparator circuit are taken from Table 7, depending on the percent of decimation band desired. The control signals, CLK and $\overline{\text{CLK}}$ sample at a 10KHz rate. To quantify the effectiveness of the decimation band in eliminating the encoding errors, the width is varied from 0% of the LSB code width (no decimation) to 15% of the LSB code width. Figure 44 shows the steady state transfer function of the SNS ADC with no decimation. Figure 45, 46, and 47 shows the transfer function for a decimation width equal to 5%, 10% and 15% of the LSB code width respectively. Note the reduction in the number of errors from Figure 44. Also note that in Figure 47 the encoding errors are all removed. The performance of the decimation band is summarized in Figure 48. As the decimation width increases, the number of encoding errors decreases. It is important to note in this simulation that the decimation band performance is a reflection on how well the crossing points are lined up across each channel. Also, note that the simulation results

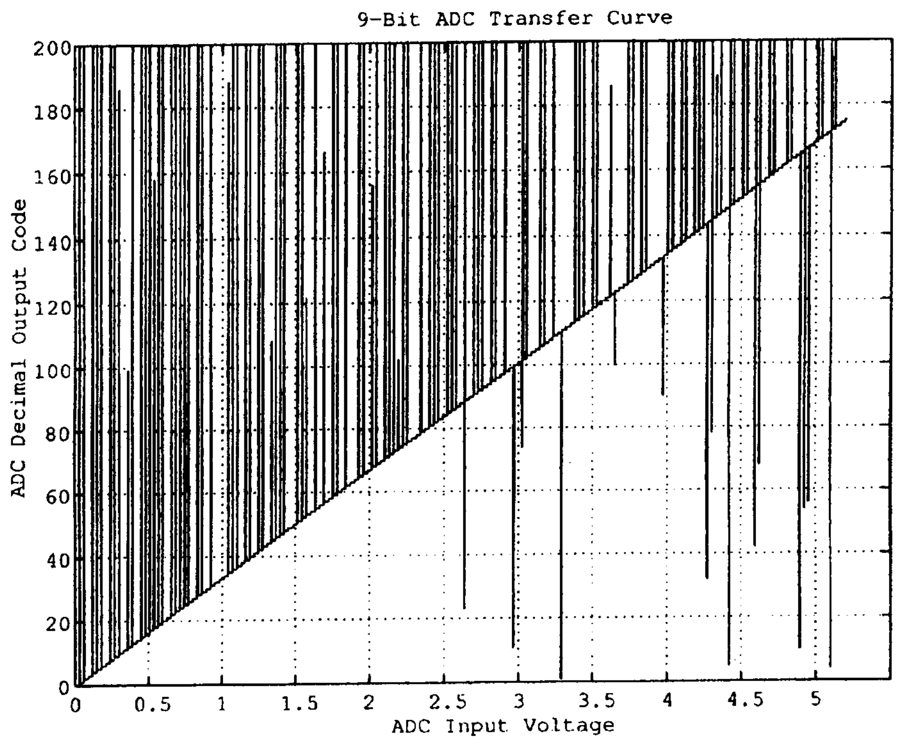


Figure 44. Steady State Transfer Function for Ideal SNS ADC.

(Decimation Width = 0% of LSB)

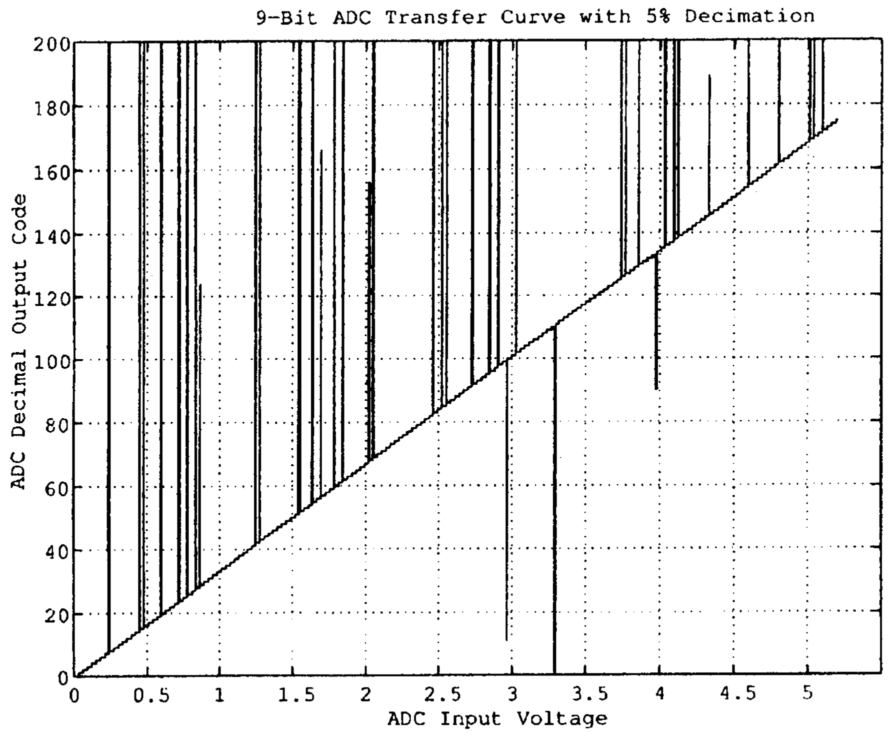


Figure 45. Steady State Transfer Function for Ideal SNS ADC.
(Decimation Width = 5% of LSB)

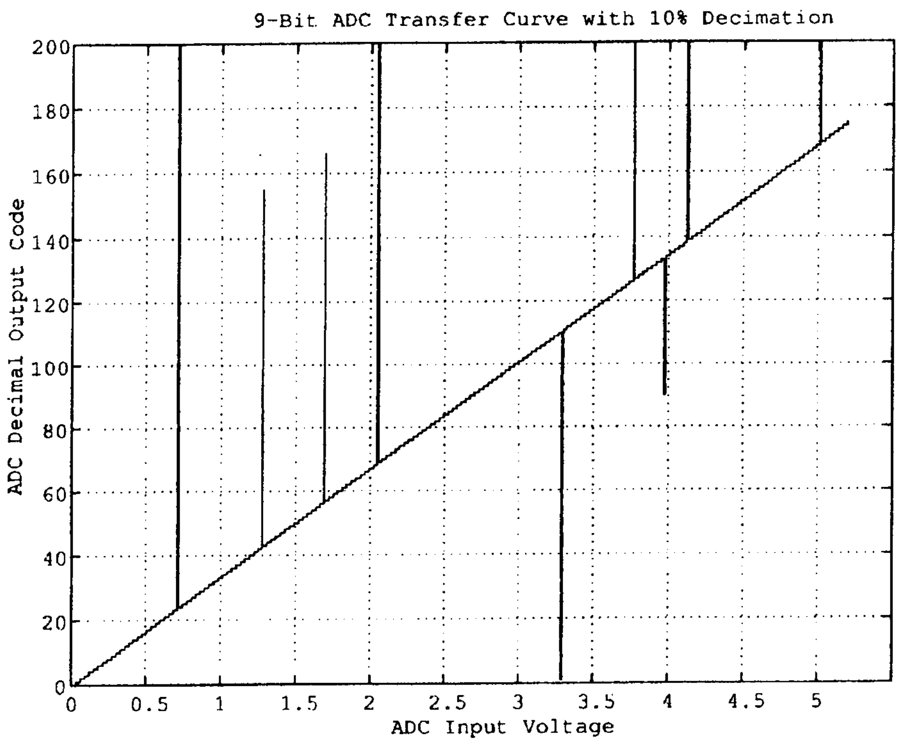


Figure 46. Steady State Transfer Function for Ideal SNS ADC.
(Decimation Width = 10% of LSB)

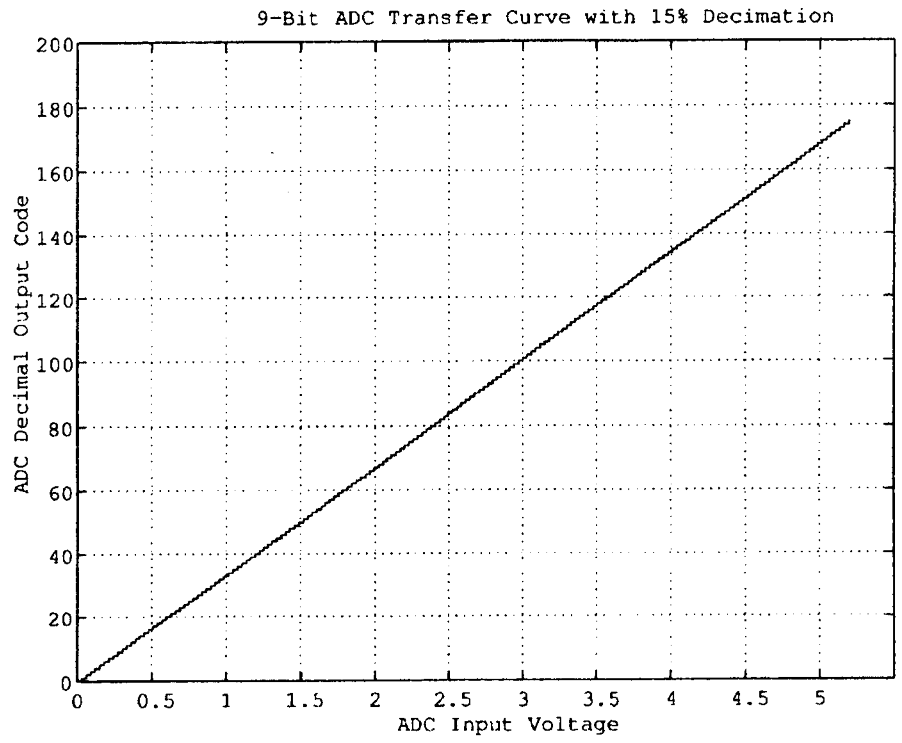


Figure 47. Steady State Transfer Function for Ideal SNS ADC.
(Decimation Width =15% of LSB)

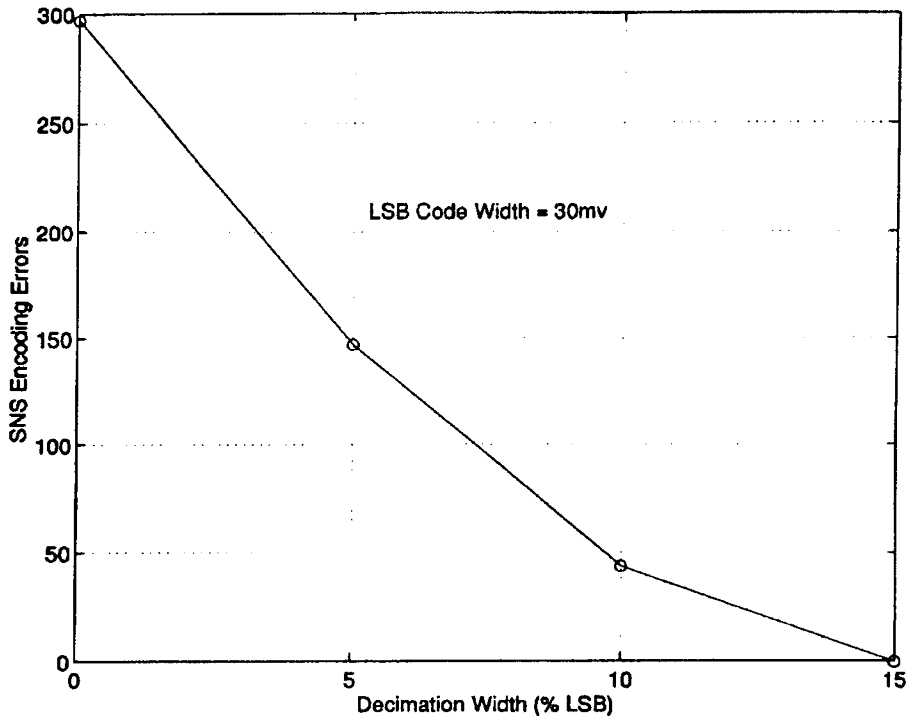


Figure 48. Number of Encoding Errors as a Function of Decimation Width.

are based on an ideal folding waveform as the input to the comparator groups.

Finally, the analysis of the decimation of encoding errors utilizing the optimum SNS preprocessing architecture is examined. The folding circuit and comparator group for each moduli is extracted and an executable spice file is created. A transient analysis of the circuit is performed by applying a linear ramp $V_{in}$ of 0.0 to 3.0 volts to the input of the folding circuit (see Figure 3). The threshold voltage $V_t$ for each comparator circuit are taken from Table 6 and Table 7. The control signals remain the same as in the previous simulation. Due to the long simulation time required of HSPICE to analyze both analog and digital circuits simultaneously, only a decimation width equal to 20% of the LSB is examined. Figure 49 shows the steady state transfer function of the actual SNS ADC with no decimation. Figure 50 shows the transfer function for a decimation width equal to 20% of the LSB code width. Note the reduction in the number of errors from Figure 49. Again, it is important to note in this simulation that the decimation band performance is a reflection on how well the crossing points are lined up across each channel and how well the folding waveform is constructed. Part A of Appendix K is the MATLAB m-file code written to simulate the modulus PLAs and the final PLA. Note that this m-file code contains only a partial listing of the 616 possible output combinations.

## G. ADC LINEARITY ERRORS

Linearity errors occur in an ADC when the actual transfer function deviates from the ideal transfer function. Due to the width of the decimation band, linearity errors are introduced. Figure 51 compares the transfer function of the ideal 9-bit ADC with the

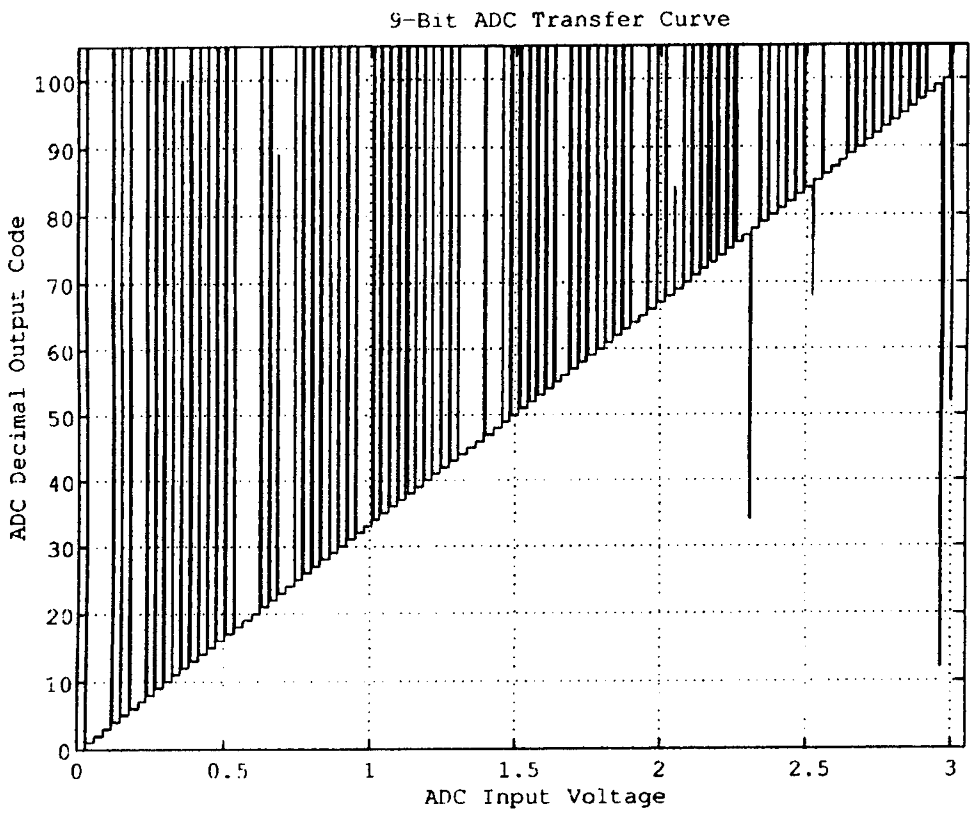


Figure 49. Steady State Transfer Function for Actual SNS ADC.
(Decimation Width = 0% of LSB)

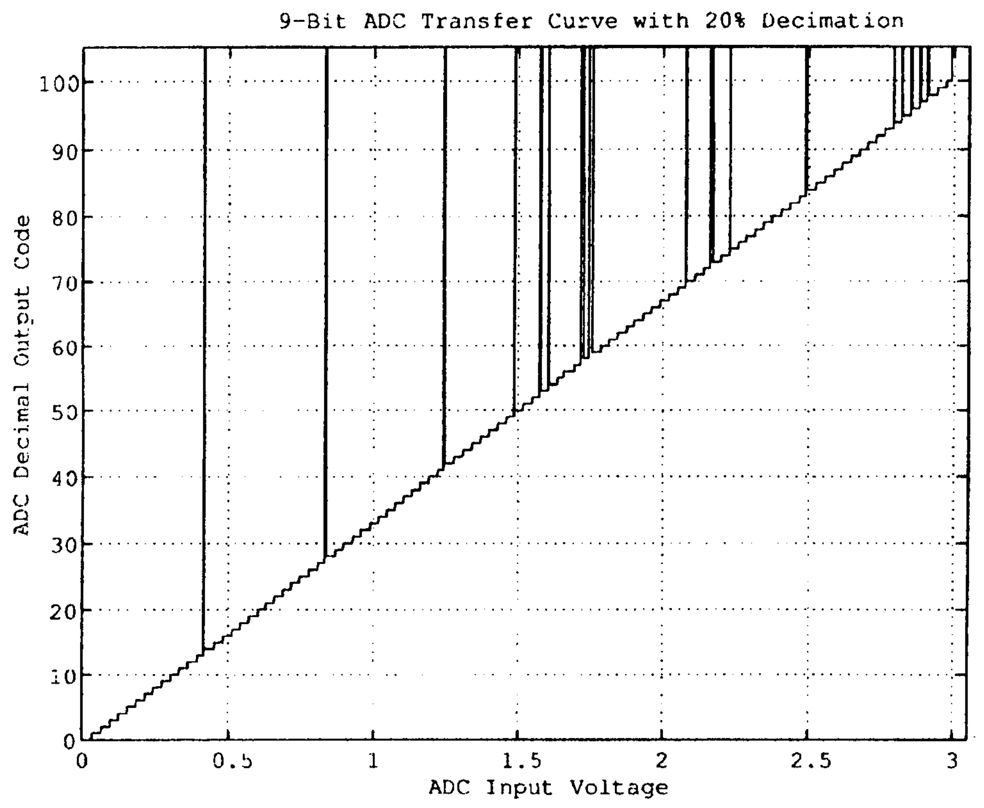


Figure 50. Steady State Transfer Function for Actual SNS ADC.
(Decimation Width = 20% of LSB)

transfer function of the actual 9-bit SNS ADC that has linearity errors. By measuring the maximum width of this linearity error throughout the dynamic range, the linearity error can be expressed as a percentage of the LSB. In this example, with the decimation band equal to 15% of the LSB, the linearity error is calculated to be 1.1% of the LSB. However, note that this result is a function of the sampling rate. An increase sampling rate would decrease the linearity error. Part B of Appendix K is the MATLAB m-file code written to compare the actual and ideal folding ADC transfer curve.

## H. POWER ESTIMATION

HSPICE can compute the maximum and average power dissipated or stored in a circuit. The *.measure* statement (found at the end of the **.spice** file) prints user-defined electrical specifications of a circuit in the **.mt#** file. The format used in this application is given below:

*.meas* avg_power avg power

*.meas* max_power max power

The first statement invokes HSPICE to measure the average power (avg_power) and report the results in the **.mt#** file under the output variable name avg power. The second statement invokes HSPICE to measure the maximum power of the circuit and report the results in the **.mt#** file under the output variable name max power. Table 15 tabulates the results of the average and maximum power consumption for each circuit discussed in this thesis.

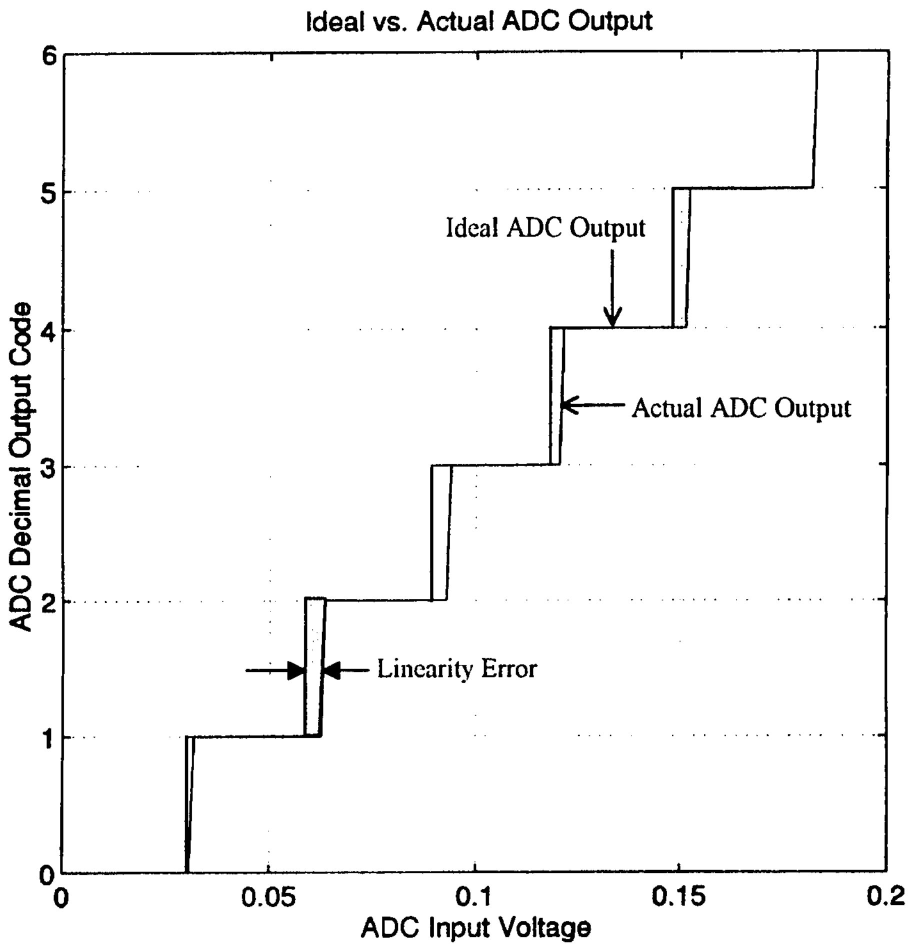


Figure 51. Linearity Error.

| Circuit | Average Power Consumed | Maximum Power Consumed |
|---|---|---|
| Comp. Group 7 | 16.82 mW | 19.61 mW |
| Comp. Group 8 | 7.35 mW | 8.76 mW |
| Comp. Group 11 | 13.57 mW | 15.66 mW |
| Parity Circuit | 166.06 μW | 7.40 mW |
| Sample and Hold Circuit | 8.42 μW | 2.51 mW |
| Mod. 7 PLA | 2.05 mW | 10.07 mW |
| Mod. 8 PLA | 2.45 mW | 12.15 mW |
| Mod. 11 PLA | 3.25 mW | 17.71 mW |
| Final PLA | 145.20 mW | 276.00 mW |
| Digital Total | 190.86 mW | 369.87 mW |
| Folding Circuit (from [Ref. 15]) | 11.25 W | 17.40 W |
| Total ADC | 11.44 W | 17.76 W |

Table 15. Average and Maximum Power Consumption.

# VII. FABRICATION

## A. SNS 9-BIT FOLDING ADC VLSI ASIC

Originally, this thesis envisioned fabricating an entire SNS 9-bit folding ADC based on the original design in Figure 2. A CMOS VLSI layout is produced consisting of both the SNS analog preprocessing architecture utilizing modulus 7, 8 and 11 [Ref. 15] and the digital architecture discussed in Chapter V. Figure 52 is the floor plan used for this layout and Figure 53 is the CMOS implementation. The final size of the CMOS VLSI design will fit on a small chip size of 4.6mm by 6.8mm.

Prior to the chip being submitted for fabrication by MOSIS using the ORBIT 2μ CMOS N-well process it was determined that fabricating the entire SNS 9-bit folding ADC would not be economically feasible without further testing and research. The VLSI SNS analog preprocessing folding circuit was evaluated and simulated in earlier research [Ref. 15]. Further research will document the test results of both the SNS folding circuit VLSI ASIC and the SNS digital VLSI ASIC.

## B. DIGITAL CIRCUIT VLSI ASIC

Since the entire SNS 9-bit folding ADC will not be fabricated, a CMOS VLSI layout based only on the SNS digital architecture is presented. In order to fabricate all three channels, two tiny chip with dimensions of 2.22mm by 2.25mm is used. Using a chip of this size will eliminate the fabrication of the final large PLA (see Figure 52). This is not a problem since the operation of PLAs are straight forward using *MPLA* and test results

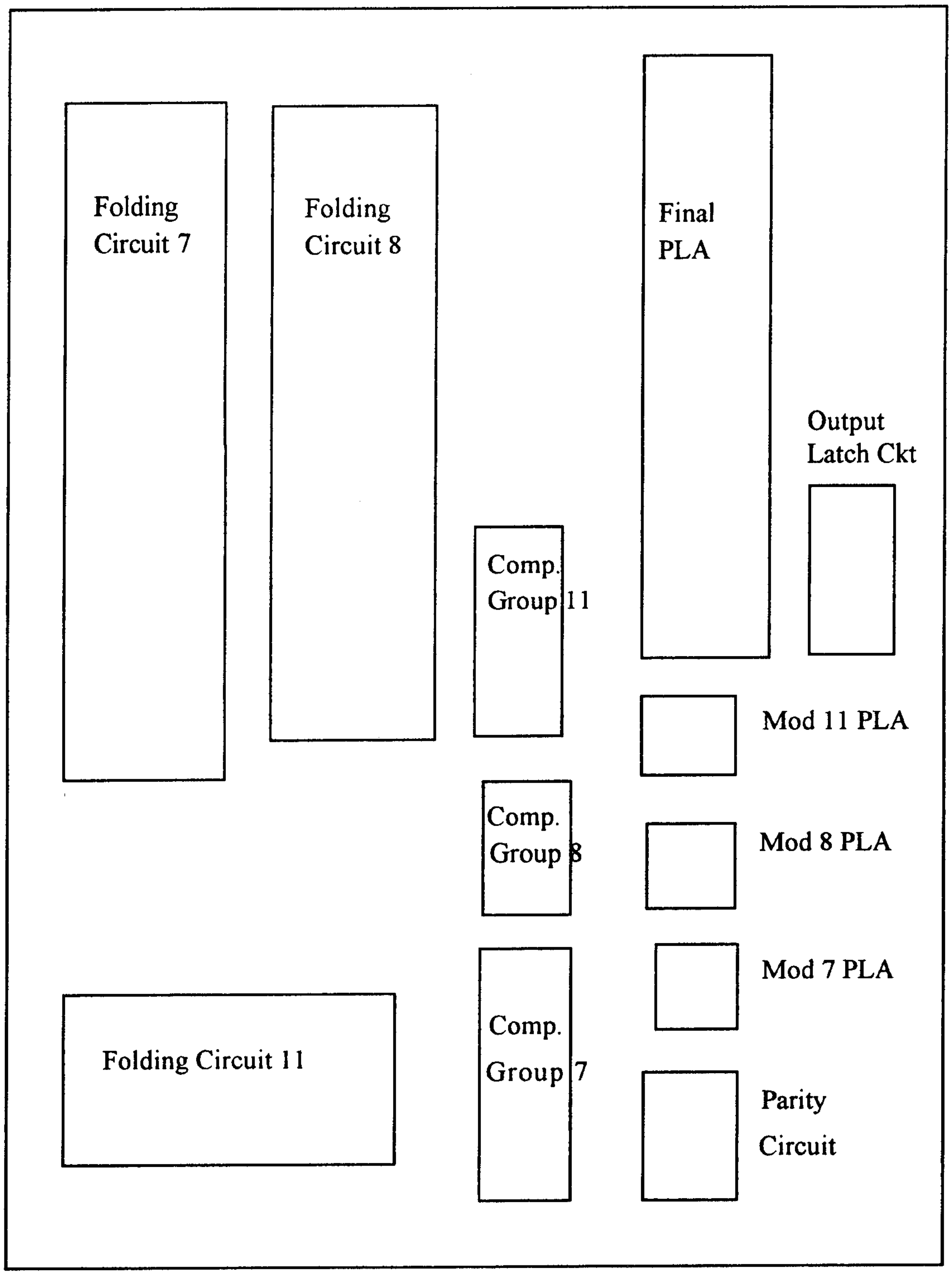


Figure 52. CMOS VLSI Architecture Floor Plan.

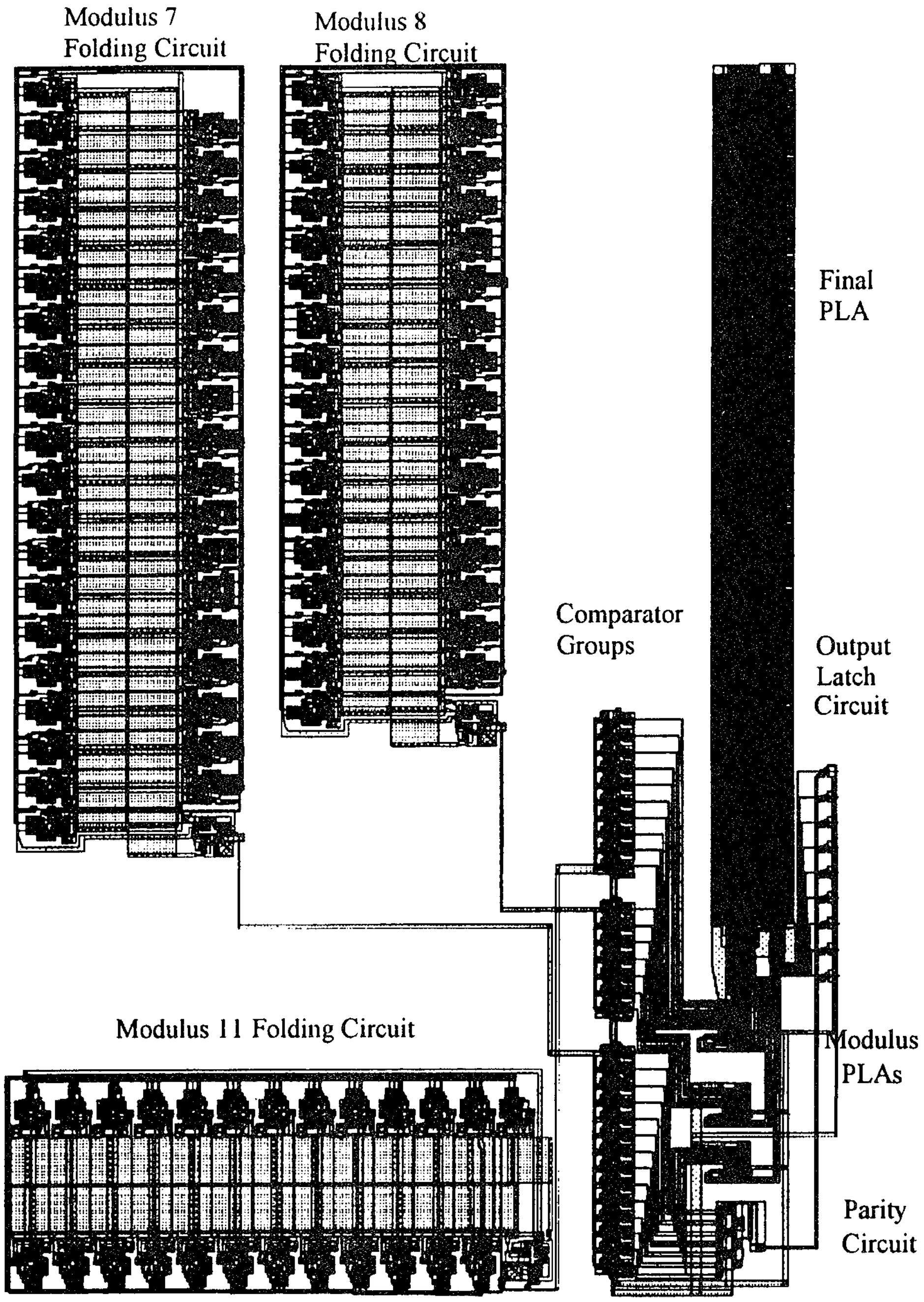


Figure 53. CMOS Implementation of Floor Plan.

obtained were satisfactory. One chip will consist of modulus 8 and 11 comparator groups and PLAs. The second chip will house the modulus 7 comparator group ($m_i$ - 1) and corresponding PLA and modulus 7 ($2m_i$), corresponding PLA and the parity circuit. Both chips will have a common interconnect provided between the individual circuits for $V_{dd}$, $V_{ss}$, CLK, $\overline{\text{CLK}}$, $V_{bias}$ and GND for each channel. Each channel will have their own $V_{in}$ (with the exception of chip two which will have a common input line) and the proper number of outputs are taken from the modulus PLAs. Also the appropriate number of lines for the threshold voltages $V_t$ will be available for each channel. Figure 54 shows the floor plan of the first digital chip (modulus 8 and 11) and Figure 55 is the floor plan for the second digital chip (modulus 7). The actual CMOS layout fabricated are shown in Figure 56 and 57 respectively.

Both chips were checked by the design rule checker (DRC) of Magic to verify the correct layout. Also, they were extracted and simulated to ensure proper functional operation. Once the layouts were confirmed to be correct, they were submitted to MOSIS for fabrication. The ORBIT 2μ CMOS N-well double-metal, double-poly process is used based on the digital nature of the layout and the fabrication schedule. Four chips of each design were fabricated and received from MOSIS packaged in a 40 pin dual in-line package (DIP).

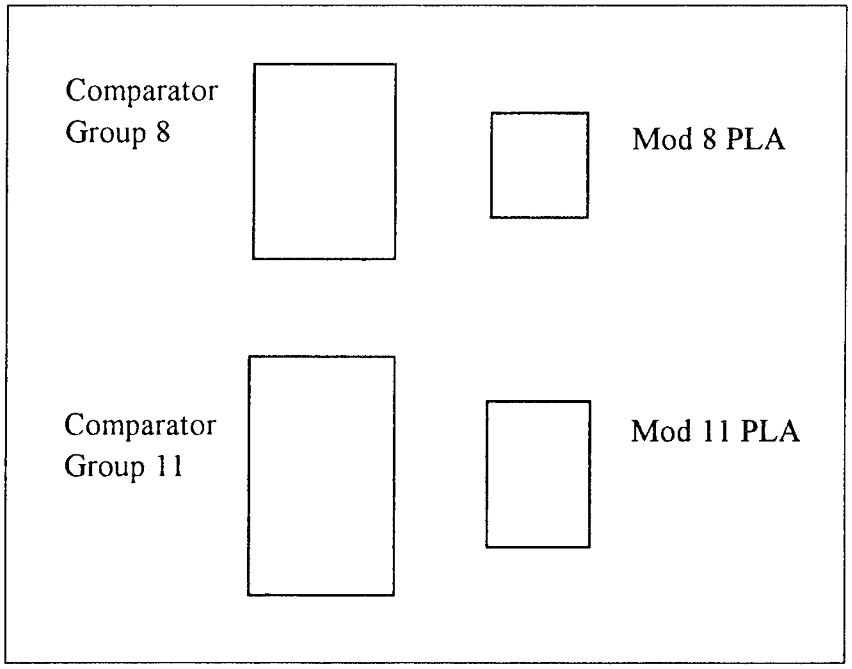


Figure 54. Digital Chip 1 Floor Plan.

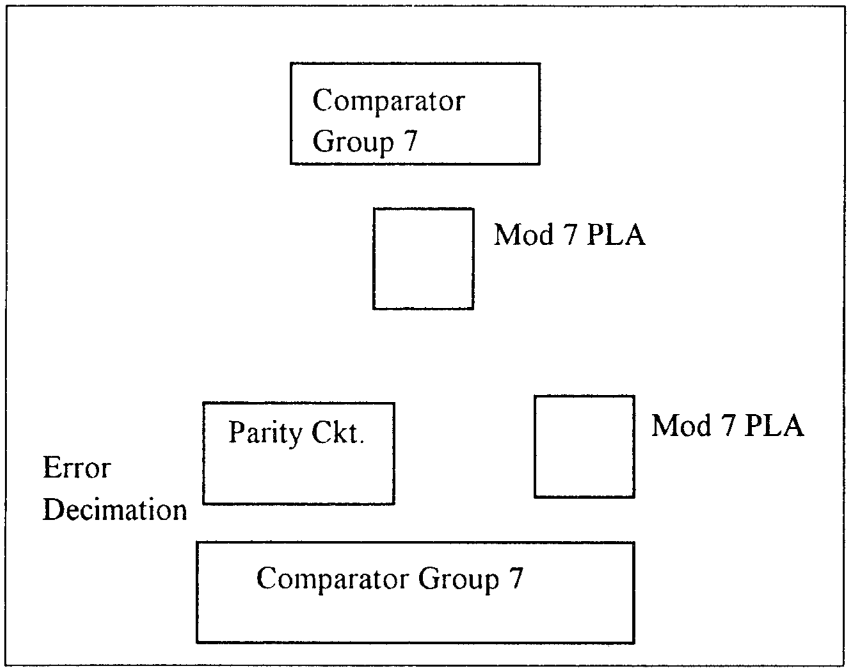


Figure 55. Digital Chip 2 Floor Plan.

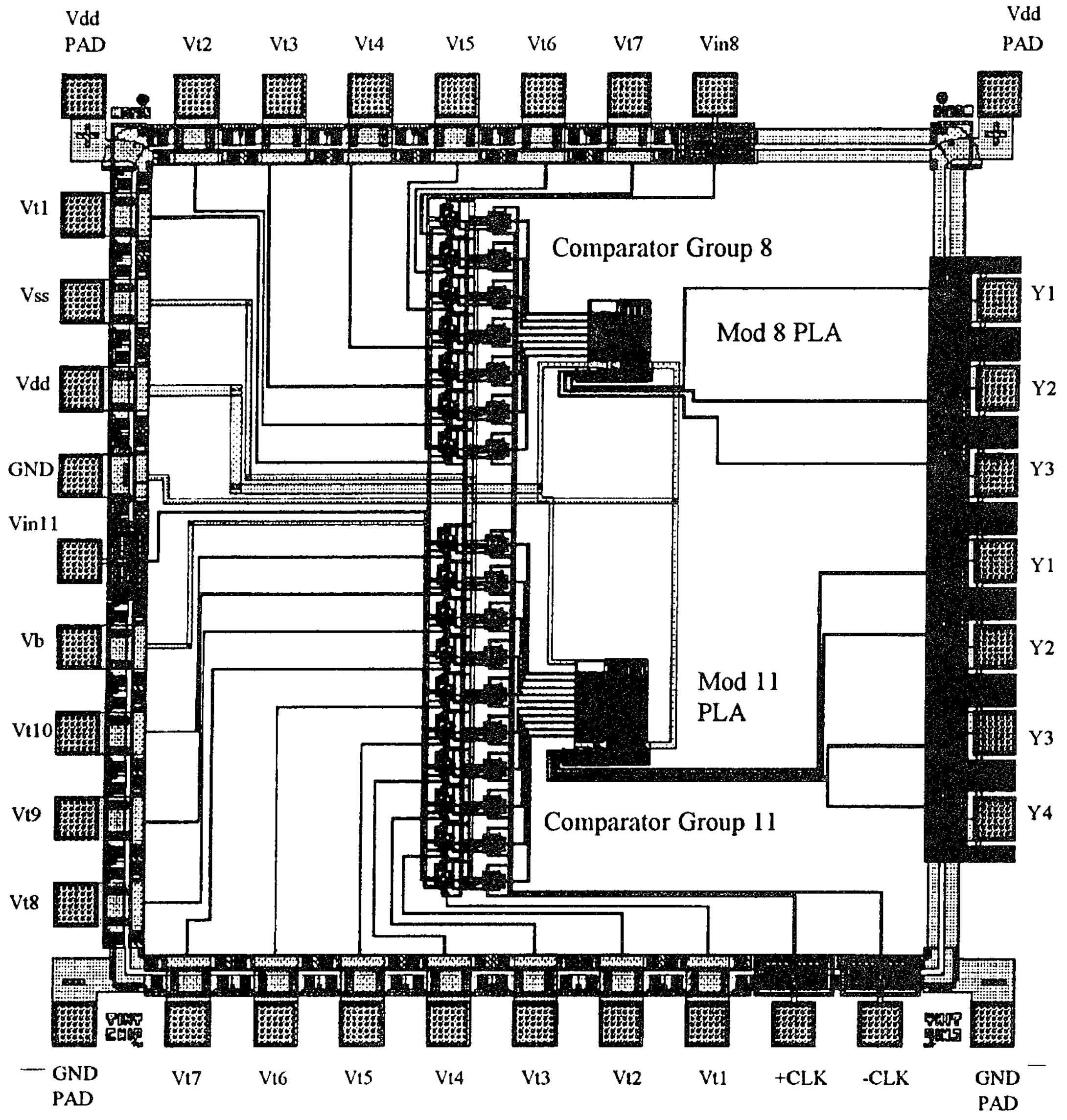


Figure 56. CMOS Implementation of Digital Chip 1.

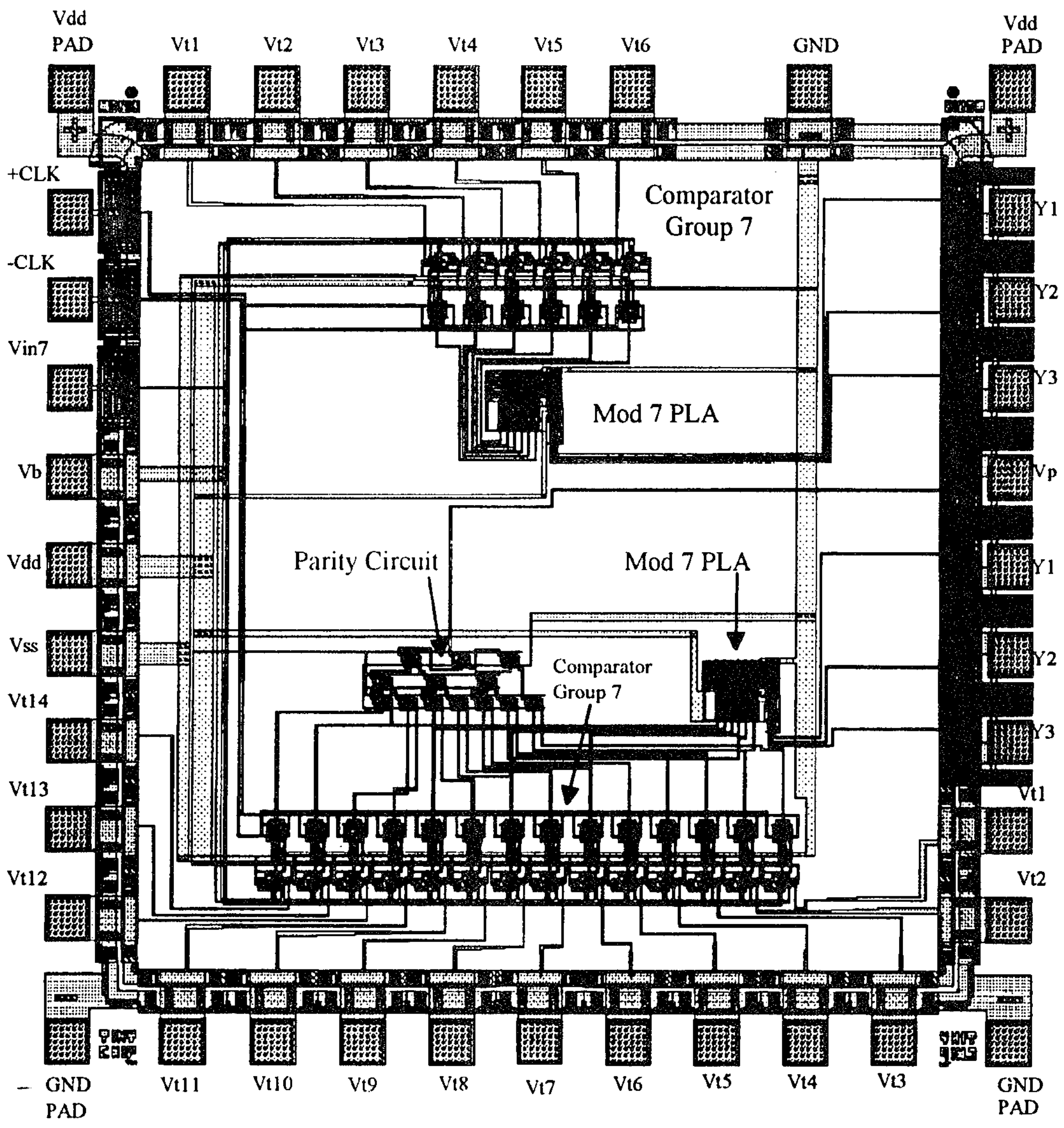


Figure 57. CMOS Implementation of Digital Chip 2.

# VIII. CONCLUSIONS, LIMITATIONS AND RECOMMENDATIONS

## A. CONCLUSIONS

This thesis examines a SNS definition that considerably extends the dynamic range of the SNS folding ADC. The major focus has been to examine the effectiveness of the decimation band in eliminating the encoding errors that occur when the folded input sample lies at one of the code transition points. This thesis also investigates the feasibility of implementing and fabricating an error detection scheme used to eliminate the encoding errors of a 2μ CMOS SNS folding ADC. All circuits in the design are successfully implemented in VLSI using the CAD tool Magic. Simulations are conducted to verify the functional operation and to confirm proper layout prior to the fabrication process. An Application Specific Integrated Chip (ASIC) has been fabricated and awaits testing to confirm the results of simulation completed.

## B. LIMITATIONS

Several limitations of the design are brought forth during the course of this work. It is found that the modulus comparator circuit is extremely sensitive to the bias voltage applied to the differential amplifier (see Figure 15). An incorrect bias voltage applied to the circuit will cause the comparator to turn ON or OFF at a different threshold voltage than the desired threshold voltage. This is due to transistors M1 and M2 not operating in the saturation region. The limitation is corrected by including the ability to adjust the bias voltage to ensure the differential amplifier pair (M1 and M2) are operating in the saturation region.

A second limitation occurred during the implementation of the original latch circuit. Though the circuit is simple, it did not lend itself well to the VLSI implementation of a folding ADC. One of the main objectives in the current research of converters has been in the reduction of the die area. The original latch circuit performed well during simulation, however it failed during the VLSI implementation due to the enormous real estate required of the capacitor in the latch circuit. The limitation is corrected by re-designing the latch circuit utilizing a clocked D-type flip-flop. Also, the awareness of VLSI layout and implementation can further reduce problems early in the design and simulation phase of a circuit.

A third limitation consisted of the elimination of the encoding errors. Originally a clocked PLA was designed for the final PLA to convert the modulus PLA inputs to a binary format output which represents the analog input signal. However, it soon became apparent the die area would be larger than desired. Therefore, an output latch circuit consisting of *N*-number of D flip-flops were connected to each of the final PLA outputs. Utilizing the output of the parity circuit for the control signals, the output signal of the ADC can be controlled.

The forth limitation is the power consumption of the entire SNS 9-bit folding ADC. The digital portion of the design dissipated on the average of 191 milliwatts, whereas the folding preprocessing circuits consumed on the average of 11 watts. Though the circuits are designed for this high power consumption, the true test will come during the actual testing of the fabricated chip.

## C. RECOMMENDATIONS

Future research into the SNS folding ADCs should be concentrated in the area of improving the folding circuits and comparator groups. The research efforts in the design of the folding circuit should be concentrated in the areas of fabrication parameters, increased frequency response and lower power consumption. By improving the folding circuit, fewer encoding errors are likely to occur while recombining the moduli outputs. The research efforts in the design of the comparator circuit should be dedicated toward ensuring that the differential amplifier remains in the saturation region under all circumstances. In its present state, the design of the comparator circuit is very sensitive to the applied bias voltage. That is, the more precise the bias voltage, the better are the alignment of the transition points. Also note that the decimation band performance is a reflection on how well the transition points are lined up across each channel. Lining up these transition points more accurately can further reduce the required decimation width. Further research should include the use of 0.8μ CMOS N-well technology as well as the use of other types of semiconductor materials. Based on favorable results from the folding circuit design [Ref. 15], the SNS analog preprocessing architecture can be coupled to this research to create a SNS 9-bit folding ADC which will have an increased dynamic range, decreased die area and power consumption.

# APPENDIX A. MICROSIM'S DESIGN CENTER FILES

## A. .CIR FILE

M4.cir

```
*C:\msim53\jeff\m4.sch
*Schematic Version 5.3 - January 1993
*Mon Sep 12  09:53:43  1994
**Analysis setup**
.DC LIN V_V60 0.0 1.600 0.0001
.OP
.LIB MOD4.lib
*From [Schematic Netlist] section of msim.ini:
.lib nom.lib
.lib dparts.lib
.lib myparts.lib
.INC "C:\MSIM53\JEFF\M4.net"
.INC "C:\MSIM53\JEFF\M4.als"
.probe/CSDF V(V-V60) VE(Q_Q116) VC(Q_Q154) VC(Q_Q156) VC(Q_Q158)

.END
```

## B. .ALS FILE

M4.ALS

```
*Schematic Aliases*

.ALIASES/CSDF
R_R1                R1 (1=vc1 2=$N_0001 )
  .                 .
  .                 .
```

# APPENDIX B. MATLAB M-FILE CODE FOR 7-BIT ADC

```
% plaed40.m
% Error Detection Decoding with Sampling Period = .4mv
% Decimation width is varied from 0% to 40% of LSB

clear
clg

m4ed40
m54data
m74data

start=1;
stop=max(size(Vmod4ed));


% Converting Modulus 4, 5, 7 Comparator Outputs to zeros & ones

t4ed=Vmod4ed(:,3:10)>1;
t5=Vmod5(:,3:6)>1;
t7=Vmod7(:,3:8)>1;


% Checking Parity of Modulus 4
% This is an Exclusive OR Circuit
% This is the first stage of the circuit
        p1= ~((~t4ed(:,1) & ~t4ed(:,2))|(t4ed(:,1) & t4ed(:,2)));

        p2= ~((~t4ed(:,3) & ~t4ed(:,4))|(t4ed(:,3) & t4ed(:,4)));

        p3= ~((~t4ed(:,5) & ~t4ed(:,6))|(t4ed(:,5) & t4ed(:,6)));

        p4= ~((~t4ed(:,7) & ~t4ed(:,8))|(t4ed(:,7) & t4ed(:,8)));

% This is the second stage of the circuit
        p5= ~((~p1 & ~p2)|(p1 & p2));

        p6= ~((~p3 & ~p4)|(p3 & p4));

% This is the final stage of the parity check circuit
        Vpar= ~((~p5 & ~p6)|(p5 & p6));
```

```
% Converting Modulus 4 Thermometer Output to decimal values
        mod4 = t4ed(:,3) + t4ed(:,5) + t4ed(:,7);

% Converting Modulus 5 Thermometer Output to decimal values
        mod5 = t5(:,1) + t5(:,2) + t5(:,3) + t5(:,4);

% Converting Modulus 7 Thermometer Output to decimal values
        mod7 = t7(:,1) + t7(:,2) + t7(:,3) + t7(:,4) + t7(:,5) + t7(:,6);


% Converting Decimal Equivalent to Digital Output
start=1;
stop=max(size(mod4));
for i=start:stop
        if (mod4(i,1)==0 & mod5(i,1)==0 & mod7(i,1)==0)
         out(i,1)=0;
        elseif (mod4(i,1)==1 & mod5(i,1)==1 & mod7(i,1)==1)
         out(i,1)=1;
        elseif (mod4(i,1)==2 & mod5(i,1)==2 & mod7(i,1)==2)
         out(i,1)=2;
        elseif (mod4(i,1)==3 & mod5(i,1)==3 & mod7(i,1)==3)
         out(i,1)=3;
        elseif (mod4(i,1)==3 & mod5(i,1)==4 & mod7(i,1)==4)
         out(i,1)=4;
        elseif (mod4(i,1)==2 & mod5(i,1)==4 & mod7(i,1)==5)
         out(i,1)=5;
        elseif (mod4(i,1)==1 & mod5(i,1)==3 & mod7(i,1)==6)
         out(i,1)=6;
        elseif (mod4(i,1)==0 & mod5(i,1)==2 & mod7(i,1)==6)
         out(i,1)=7;
        elseif (mod4(i,1)==0 & mod5(i,1)==1 & mod7(i,1)==5)
         out(i,1)=8;
        elseif (mod4(i,1)==1 & mod5(i,1)==0 & mod7(i,1)==4)
         out(i,1)=9;
        elseif (mod4(i,1)==2 & mod5(i,1)==0 & mod7(i,1)==3)
         out(i,1)=10;
        elseif (mod4(i,1)==3 & mod5(i,1)==1 & mod7(i,1)==2)
         out(i,1)=11;
        elseif (mod4(i,1)==3 & mod5(i,1)==2 & mod7(i,1)==1)
         out(i,1)=12;
        elseif (mod4(i,1)==2 & mod5(i,1)==3 & mod7(i,1)==0)
         out(i,1)=13;
        elseif (mod4(i,1)==1 & mod5(i,1)==4 & mod7(i,1)==0)
         out(i,1)=14;
        elseif (mod4(i,1)==0 & mod5(i,1)==4 & mod7(i,1)==1)
```

```
  out(i,1)=15;
 elseif (mod4(i,1)==0 & mod5(i,1)==3 & mod7(i,1)==2)
  out(i,1)=16;
 elseif (mod4(i,1)==1 & mod5(i,1)==2 & mod7(i,1)==3)
  out(i,1)=17;
 elseif (mod4(i,1)==2 & mod5(i,1)==1 & mod7(i,1)==4)
  out(i,1)=18;
 elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==5)
  out(i,1)=19;
 elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==6)
  out(i,1)=20;
 elseif (mod4(i,1)==2 & mod5(i,1)==1 & mod7(i,1)==6)
  out(i,1)=21;
 elseif (mod4(i,1)==1 & mod5(i,1)==2 & mod7(i,1)==5)
  out(i,1)=22;
 elseif (mod4(i,1)==0 & mod5(i,1)==3 & mod7(i,1)==4)
  out(i,1)=23;
 elseif (mod4(i,1)==0 & mod5(i,1)==4 & mod7(i,1)==3)
  out(i,1)=24;
 elseif (mod4(i,1)==1 & mod5(i,1)==4 & mod7(i,1)==2)
  out(i,1)=25;
 elseif (mod4(i,1)==2 & mod5(i,1)==3 & mod7(i,1)==1)
  out(i,1)=26;
 elseif (mod4(i,1)==3 & mod5(i,1)==2 & mod7(i,1)==0)
  out(i,1)=27;
 elseif (mod4(i,1)==3 & mod5(i,1)==1 & mod7(i,1)==0)
  out(i,1)=28;
 elseif (mod4(i,1)==2 & mod5(i,1)==0 & mod7(i,1)==1)
  out(i,1)=29;
 elseif (mod4(i,1)==1 & mod5(i,1)==0 & mod7(i,1)==2)
  out(i,1)=30;
 elseif (mod4(i,1)==0 & mod5(i,1)==1 & mod7(i,1)==3)
  out(i,1)=31;
 elseif (mod4(i,1)==0 & mod5(i,1)==2 & mod7(i,1)==4)
  out(i,1)=32;
 elseif (mod4(i,1)==1 & mod5(i,1)==3 & mod7(i,1)==5)
  out(i,1)=33;
 elseif (mod4(i,1)==2 & mod5(i,1)==4 & mod7(i,1)==6)
  out(i,1)=34;
 elseif (mod4(i,1)==3 & mod5(i,1)==4 & mod7(i,1)==6)
  out(i,1)=35;
 elseif (mod4(i,1)==3 & mod5(i,1)==3 & mod7(i,1)==5)
  out(i,1)=36;
 elseif (mod4(i,1)==2 & mod5(i,1)==2 & mod7(i,1)==4)
  out(i,1)=37;
```

```
elseif (mod4(i,1)==1 & mod5(i,1)==1 & mod7(i,1)==3)
 out(i,1)=38;
elseif (mod4(i,1)==0 & mod5(i,1)==0 & mod7(i,1)==2)
 out(i,1)=39;
elseif (mod4(i,1)==0 & mod5(i,1)==0 & mod7(i,1)==1)
 out(i,1)=40;
elseif (mod4(i,1)==1 & mod5(i,1)==1 & mod7(i,1)==0)
 out(i,1)=41;
elseif (mod4(i,1)==2 & mod5(i,1)==2 & mod7(i,1)==0)
 out(i,1)=42;
elseif (mod4(i,1)==3 & mod5(i,1)==3 & mod7(i,1)==1)
 out(i,1)=43;
elseif (mod4(i,1)==3 & mod5(i,1)==4 & mod7(i,1)==2)
 out(i,1)=44;
elseif (mod4(i,1)==2 & mod5(i,1)==4 & mod7(i,1)==3)
 out(i,1)=45;
elseif (mod4(i,1)==1 & mod5(i,1)==3 & mod7(i,1)==4)
 out(i,1)=46;
elseif (mod4(i,1)==0 & mod5(i,1)==2 & mod7(i,1)==5)
 out(i,1)=47;
elseif (mod4(i,1)==0 & mod5(i,1)==1 & mod7(i,1)==6)
 out(i,1)=48;
elseif (mod4(i,1)==1 & mod5(i,1)==0 & mod7(i,1)==6)
 out(i,1)=49;
elseif (mod4(i,1)==2 & mod5(i,1)==0 & mod7(i,1)==5)
 out(i,1)=50;
elseif (mod4(i,1)==3 & mod5(i,1)==1 & mod7(i,1)==4)
 out(i,1)=51;
elseif (mod4(i,1)==3 & mod5(i,1)==2 & mod7(i,1)==3)
 out(i,1)=52;
elseif (mod4(i,1)==2 & mod5(i,1)==3 & mod7(i,1)==2)
 out(i,1)=53;
elseif (mod4(i,1)==1 & mod5(i,1)==4 & mod7(i,1)==1)
 out(i,1)=54;
elseif (mod4(i,1)==0 & mod5(i,1)==4 & mod7(i,1)==0)
 out(i,1)=55;
elseif (mod4(i,1)==0 & mod5(i,1)==3 & mod7(i,1)==0)
 out(i,1)=56;
elseif (mod4(i,1)==1 & mod5(i,1)==2 & mod7(i,1)==1)
 out(i,1)=57;
elseif (mod4(i,1)==2 & mod5(i,1)==1 & mod7(i,1)==2)
 out(i,1)=58;
elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==3)
 out(i,1)=59;
elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==4)
```

```
  out(i,1)=60;
 elseif (mod4(i,1)==2 & mod5(i,1)==1 & mod7(i,1)==5)
  out(i,1)=61;
 elseif (mod4(i,1)==1 & mod5(i,1)==2 & mod7(i,1)==6)
  out(i,1)=62;
 elseif (mod4(i,1)==0 & mod5(i,1)==3 & mod7(i,1)==6)
  out(i,1)=63;
 elseif (mod4(i,1)==0 & mod5(i,1)==4 & mod7(i,1)==5)
  out(i,1)=64;
 elseif (mod4(i,1)==1 & mod5(i,1)==4 & mod7(i,1)==4)
  out(i,1)=65;
 elseif (mod4(i,1)==2 & mod5(i,1)==3 & mod7(i,1)==3)
  out(i,1)=66;
 elseif (mod4(i,1)==3 & mod5(i,1)==2 & mod7(i,1)==2)
  out(i,1)=67;
 elseif (mod4(i,1)==3 & mod5(i,1)==1 & mod7(i,1)==1)
  out(i,1)=68;
 elseif (mod4(i,1)==2 & mod5(i,1)==0 & mod7(i,1)==0)
  out(i,1)=69;
 elseif (mod4(i,1)==1 & mod5(i,1)==0 & mod7(i,1)==0)
  out(i,1)=70;
 elseif (mod4(i,1)==0 & mod5(i,1)==1 & mod7(i,1)==1)
  out(i,1)=71;
 elseif (mod4(i,1)==0 & mod5(i,1)==2 & mod7(i,1)==2)
  out(i,1)=72;
 elseif (mod4(i,1)==1 & mod5(i,1)==3 & mod7(i,1)==3)
  out(i,1)=73;
 elseif (mod4(i,1)==2 & mod5(i,1)==4 & mod7(i,1)==4)
  out(i,1)=74;
 elseif (mod4(i,1)==3 & mod5(i,1)==4 & mod7(i,1)==5)
  out(i,1)=75;
 elseif (mod4(i,1)==3 & mod5(i,1)==3 & mod7(i,1)==6)
  out(i,1)=76;
 elseif (mod4(i,1)==2 & mod5(i,1)==2 & mod7(i,1)==6)
  out(i,1)=77;
 elseif (mod4(i,1)==1 & mod5(i,1)==1 & mod7(i,1)==5)
  out(i,1)=78;
 elseif (mod4(i,1)==0 & mod5(i,1)==0 & mod7(i,1)==4)
  out(i,1)=79;
 elseif (mod4(i,1)==0 & mod5(i,1)==0 & mod7(i,1)==3)
  out(i,1)=80;
 elseif (mod4(i,1)==1 & mod5(i,1)==1 & mod7(i,1)==2)
  out(i,1)=81;
 elseif (mod4(i,1)==2 & mod5(i,1)==2 & mod7(i,1)==1)
  out(i,1)=82;
```

```
elseif (mod4(i,1)==3 & mod5(i,1)==3 & mod7(i,1)==0)
 out(i,1)=83;
elseif (mod4(i,1)==3 & mod5(i,1)==4 & mod7(i,1)==0)
 out(i,1)=84;
elseif (mod4(i,1)==2 & mod5(i,1)==4 & mod7(i,1)==1)
 out(i,1)=85;
elseif (mod4(i,1)==1 & mod5(i,1)==3 & mod7(i,1)==2)
 out(i,1)=86;
elseif (mod4(i,1)==0 & mod5(i,1)==2 & mod7(i,1)==3)
 out(i,1)=87;
elseif (mod4(i,1)==0 & mod5(i,1)==1 & mod7(i,1)==4)
 out(i,1)=88;
elseif (mod4(i,1)==1 & mod5(i,1)==0 & mod7(i,1)==5)
 out(i,1)=89;
elseif (mod4(i,1)==2 & mod5(i,1)==0 & mod7(i,1)==6)
 out(i,1)=90;
elseif (mod4(i,1)==3 & mod5(i,1)==1 & mod7(i,1)==6)
 out(i,1)=91;
elseif (mod4(i,1)==3 & mod5(i,1)==2 & mod7(i,1)==5)
 out(i,1)=92;
elseif (mod4(i,1)==2 & mod5(i,1)==3 & mod7(i,1)==4)
 out(i,1)=93;
elseif (mod4(i,1)==1 & mod5(i,1)==4 & mod7(i,1)==3)
 out(i,1)=94;
elseif (mod4(i,1)==0 & mod5(i,1)==4 & mod7(i,1)==2)
 out(i,1)=95;
elseif (mod4(i,1)==0 & mod5(i,1)==3 & mod7(i,1)==1)
 out(i,1)=96;
elseif (mod4(i,1)==1 & mod5(i,1)==2 & mod7(i,1)==0)
 out(i,1)=97;
elseif (mod4(i,1)==2 & mod5(i,1)==1 & mod7(i,1)==0)
 out(i,1)=98;
elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==1)
 out(i,1)=99;
elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==2)
 out(i,1)=100;
elseif (mod4(i,1)==2 & mod5(i,1)==1 & mod7(i,1)==3)
 out(i,1)=101;
elseif (mod4(i,1)==1 & mod5(i,1)==2 & mod7(i,1)==4)
 out(i,1)=102;
elseif (mod4(i,1)==0 & mod5(i,1)==3 & mod7(i,1)==5)
 out(i,1)=103;
elseif (mod4(i,1)==0 & mod5(i,1)==4 & mod7(i,1)==6)
 out(i,1)=104;
elseif (mod4(i,1)==1 & mod5(i,1)==4 & mod7(i,1)==6)
```

```
 out(i,1)=105;
elseif (mod4(i,1)==2 & mod5(i,1)==3 & mod7(i,1)==5)
 out(i,1)=106;
elseif (mod4(i,1)==3 & mod5(i,1)==2 & mod7(i,1)==4)
 out(i,1)=107;
elseif (mod4(i,1)==3 & mod5(i,1)==1 & mod7(i,1)==3)
 out(i,1)=108;
elseif (mod4(i,1)==2 & mod5(i,1)==0 & mod7(i,1)==2)
 out(i,1)=109;
elseif (mod4(i,1)==1 & mod5(i,1)==0 & mod7(i,1)==1)
 out(i,1)=110;
elseif (mod4(i,1)==0 & mod5(i,1)==1 & mod7(i,1)==0)
 out(i,1)=111;
elseif (mod4(i,1)==0 & mod5(i,1)==2 & mod7(i,1)==0)
 out(i,1)=112;
elseif (mod4(i,1)==1 & mod5(i,1)==3 & mod7(i,1)==1)
 out(i,1)=113;
elseif (mod4(i,1)==2 & mod5(i,1)==4 & mod7(i,1)==2)
 out(i,1)=114;
elseif (mod4(i,1)==3 & mod5(i,1)==4 & mod7(i,1)==3)
 out(i,1)=115;
elseif (mod4(i,1)==3 & mod5(i,1)==3 & mod7(i,1)==4)
 out(i,1)=116;
elseif (mod4(i,1)==2 & mod5(i,1)==2 & mod7(i,1)==5)
 out(i,1)=117;
elseif (mod4(i,1)==1 & mod5(i,1)==1 & mod7(i,1)==6)
 out(i,1)=118;
elseif (mod4(i,1)==0 & mod5(i,1)==0 & mod7(i,1)==6)
 out(i,1)=119;
elseif (mod4(i,1)==0 & mod5(i,1)==0 & mod7(i,1)==5)
 out(i,1)=120;
elseif (mod4(i,1)==1 & mod5(i,1)==1 & mod7(i,1)==4)
 out(i,1)=121;
elseif (mod4(i,1)==2 & mod5(i,1)==2 & mod7(i,1)==3)
 out(i,1)=122;
elseif (mod4(i,1)==3 & mod5(i,1)==3 & mod7(i,1)==2)
 out(i,1)=123;
elseif (mod4(i,1)==3 & mod5(i,1)==4 & mod7(i,1)==1)
 out(i,1)=124;
elseif (mod4(i,1)==2 & mod5(i,1)==4 & mod7(i,1)==0)
 out(i,1)=125;
elseif (mod4(i,1)==1 & mod5(i,1)==3 & mod7(i,1)==0)
 out(i,1)=126;
elseif (mod4(i,1)==0 & mod5(i,1)==2 & mod7(i,1)==1)
 out(i,1)=127;
```

```
        elseif (mod4(i,1)==0 & mod5(i,1)==1 & mod7(i,1)==2)
         out(i,1)=128;
        elseif (mod4(i,1)==1 & mod5(i,1)==0 & mod7(i,1)==3)
         out(i,1)=129;
        elseif (mod4(i,1)==2 & mod5(i,1)==0 & mod7(i,1)==4)
         out(i,1)=130;
        elseif (mod4(i,1)==3 & mod5(i,1)==1 & mod7(i,1)==5)
         out(i,1)=131;
        elseif (mod4(i,1)==3 & mod5(i,1)==2 & mod7(i,1)==6)
         out(i,1)=132;
        elseif (mod4(i,1)==2 & mod5(i,1)==3 & mod7(i,1)==6)
         out(i,1)=133;
        elseif (mod4(i,1)==1 & mod5(i,1)==4 & mod7(i,1)==5)
         out(i,1)=134;
        elseif (mod4(i,1)==0 & mod5(i,1)==4 & mod7(i,1)==4)
         out(i,1)=135;
        elseif (mod4(i,1)==0 & mod5(i,1)==3 & mod7(i,1)==3)
         out(i,1)=136;
        elseif (mod4(i,1)==1 & mod5(i,1)==2 & mod7(i,1)==2)
         out(i,1)=137;
        elseif (mod4(i,1)==2 & mod5(i,1)==1 & mod7(i,1)==1)
         out(i,1)=138;
        elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==0)
         out(i,1)=139;
        elseif (mod4(i,1)==3 & mod5(i,1)==0 & mod7(i,1)==0)
         out(i,1)=140;
        else
         out(i,1)=150;
        end
end

% Check the parity, if even then discard value and hold output at the % last known good value

temp=0;
stop=max(size(out));
for i=start:stop

        if (Vpar(i))            % parity is odd, keep
         errorcor(i)=out(i);
         temp=out(i);
        else                    % parity is even, discard
         errorcor(i)=temp;
        end
end
ec=errorcor';
```

```
% Compute the ideal line corresponding to the transfer curve

  p = polyfit(Vmod4ed([start,stop],1),ec([start,stop],1),1);
  yfitted = polyval(p,Vmod4ed(:,1));

%Looking for errors>1 at every step

  error = abs(ec(:,1)-yfitted);

%find returns the location of all values that meet the given criteria.

  f = find(error>1.5);

%errorcnt will contain the number of errors

  errorcnt40 = length(f);

%finding linearity errors

linerror40=(ec(:,1)-yfitted);
linearerror=max(linerror40)*100/12;

% Plot the error corrected output
plot(Vmod4ed(:,1), ec(:,1))
title('Decimation Width = 40% LSB')
ylabel('ADC Decimal Output Code')
xlabel('ADC Input Voltage')
grid
pause
print

% Plot the linearity error
plot(Vmod4ed(:,1), linerror40)
title('Decimation Width = 40% LSB')
ylabel('Linearity Errors')
xlabel('ADC Input Voltage')
grid
pause
print
```

# APPENDIX C: MPLA INPUT FILE FOR MODULUS 7, 8 AND 11

# Input files for generation of Modulus PLAs through the use of Mpla CAD Tool.

## A. MPLA Input File for Modulus 7 PLA.

```
.na pla
.ilb  X6 X5 X4 X3 X2 X1
.orb  Y3 Y2 Y1

#### PLA TRUTH TABLE  ####
.i      6
.o      3
.p      7
000000       000
000001       001
000011       010
000111       011
001111       100
011111       101
111111       110
.e
```

## B. MPLA Input File for Modulus 8 PLA.

```
.na pla
.ilb  X7 X6 X5 X4 X3 X2 X1
.orb  Y3 Y2 Y1

#### PLA TRUTH TABLE  ####
.i      7
.o      3
.p      8
0000000      000
0000001      001
0000011      010
0000111      011
0001111      100
0011111      101
0111111      110
1111111      111
.e
```

## C. MPLA Input File for Modulus 11 PLA.

```
.na pla
.ilb  X10 X9 X8 X7 X6 X5 X4 X3 X2 X1
.orb  Y4 Y3 Y2 Y1

#### PLA TRUTH TABLE  ####
.i      10
.o      4
.p      11
0000000000   0000
0000000001   0001
0000000011   0010
0000000111   0011
0000001111   0100
0000011111   0101
0000111111   0110
0001111111   0111
0011111111   1000
0111111111   1001
1111111111   1010
.e
```

# APPENDIX D: MATLAB M-FILE CODE FOR THE GENERATION OF THE FINAL PLA TRUTH TABLE

```
%This M-file generates the 616 input and output vectors required to
% generate the final PLA.

clear
format compact

% *********************Generate input Vectors  **********************
% The input vectors for the pla truth table are generated by combining
% the individual modulus vectors.

%**************************************************************************
%First the individual modulus vectors are specified and are referred as seed
%vectors.
seed7=[0 0 0
    0 0 1
    0 1 0
    0 1 1
    1 0 0
    1 0 1
    1 1 0
    1 1 0
    1 0 1
    1 0 0
    0 1 1
    0 1 0
    0 0 1
    0 0 0];

seed8=[0 0 0
     0 0 1
     0 1 0
     0 1 1
     1 0 0
     1 0 1
     1 1 0
     1 1 1
     1 1 1
     1 1 0
     1 0 1
     1 0 0
     0 1 1
     0 1 0
```

```
      0 0 1
      0 0 0];

 seed11=[0 0 0 0
       0 0 0 1
       0 0 1 0
       0 0 1 1
       0 1 0 0
       0 1 0 1
       0 1 1 0
       0 1 1 1
       1 0 0 0
       1 0 0 1
       1 0 1 0
       1 0 1 0
       1 0 0 1
       1 0 0 0
       0 1 1 1
       0 1 1 0
       0 1 0 1
       0 1 0 0
       0 0 1 1
       0 0 1 0
       0 0 0 1
       0 0 0 0];


%The number of times each modulus seed matrix must repeat itself to reach
%the desired dynamic range is computed.
sz7=max(size(seed7));
sz8=max(size(seed8));
sz11=max(size(seed11));

n7=ceil(512/sz7);
n8=ceil(512/sz8);
n11=ceil(512/sz11);

%The individual modulus matrix is generated for the specified dynamic range.
for j=1:n7

   input7=[input7;seed7];
end

for j=1:n8
```

```
   input8=[input8;seed8];
end

for j=1:n11

   input11=[input11;seed11];
end

input7=input7(1:512,:);
input8=input8(1:512,:);
input11=input11(1:512,:);

%The SNS matrix representing the input part of the truth table is generated
%by combining the individual modulus vectors
diary on
snsv=[input11 input8 input7]
diary off

%This produces a 9-bit binary table

seedb1=[zeros(1,1);ones(1,1)];
seedb2=[zeros(2,1);ones(2,1)];
seedb3=[zeros(4,1);ones(4,1)];
seedb4=[zeros(8,1);ones(8,1)];
seedb5=[zeros(16,1);ones(16,1)];
seedb6=[zeros(32,1);ones(32,1)];
seedb7=[zeros(64,1);ones(64,1)];
seedb8=[zeros(128,1);ones(128,1)];
seedb9=[zeros(256,1);ones(256,1)];

szb1=max(size(seedb1));
szb2=max(size(seedb2));
szb3=max(size(seedb3));
szb4=max(size(seedb4));
szb5=max(size(seedb5));
szb6=max(size(seedb6));
szb7=max(size(seedb7));
szb8=max(size(seedb8));

nb1=ceil(512/szb1);
nb2=ceil(512/szb2);
nb3=ceil(512/szb3);
nb4=ceil(512/szb4);
nb5=ceil(512/szb5);
nb6=ceil(512/szb6);
```

```
nb7=ceil(512/szb7);
nb8=ceil(512/szb8);

for j=1:nb1
        outputb1=[outputb1;seedb1];
end

for j=1:nb2
        outputb2=[outputb2;seedb2];
end

for j=1:nb3
        outputb3=[outputb3;seedb3];
end

for j=1:nb4
        outputb4=[outputb4;seedb4];
end

for j=1:nb5
        outputb5=[outputb5;seedb5];
end

for j=1:nb6
        outputb6=[outputb6;seedb6];
end

for j=1:nb7
        outputb7=[outputb7;seedb7];
end

for j=1:nb8
        outputb8=[outputb8;seedb8];
end

output=[seedb9 outputb8 outputb7 outputb6 outputb5 outputb4 outputb3 outputb2 outputb1];

s=3;
for j=1:512
        sbg(j)=s;
end
sbg=sbg';
platt=[snsv sbg output]
```

# APPENDIX E: MPLA INPUT FILE FOR FINAL PLA

```
#Input file for the generation of the Final PLA using Mpla CAD Tool.

.na pla
.ilb    X10 X9 X8 X7 X6 X5 X4 X3 X2 X1
.orb    Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1

########PLA Truth Table############
#Inputs [Mod 11(X10-X7)  Mod 8(X6-X4)  Mod 7(X3-X1)]  Outputs (Y9-Y1)
.i 10
.o 9
.p 512
0000000000 000000000
0001001001 000000001
0010010010 000000010
0011011011 000000011
0100100100 000000100
0101101101 000000101
0110110110 000000110
0111111110 000000111
1000111101 000001000
1001110100 000001001
1010101011 000001010
1010100010 000001011
1001011001 000001100
1000010000 000001101
0111001000 000001110
0110000001 000001111
0101000010 000010000
0100001011 000010001
0011010100 000010010
0010011101 000010011
0001100110 000010100
0000101110 000010101
0000110101 000010110
0001111100 000010111
0010111011 000011000
0011110010 000011001
0100101001 000011010
0101100000 000011011
0110011000 000011100
0111010001 000011101
1000001010 000011110
```

| | |
|---|---|
| 1001000011 | 000011111 |
| 1010000100 | 000100000 |
| 1010001101 | 000100001 |
| 1001010110 | 000100010 |
| 1000011110 | 000100011 |
| 0111100101 | 000100100 |
| 0110101100 | 000100101 |
| 0101110011 | 000100110 |
| 0100111010 | 000100111 |
| 0011111001 | 000101000 |
| 0010110000 | 000101001 |
| 0001101000 | 000101010 |
| 0000100001 | 000101011 |
| 0000011010 | 000101100 |
| 0001010011 | 000101101 |
| 0010001100 | 000101110 |
| 0011000101 | 000101111 |
| 0100000110 | 000110000 |
| 0101001110 | 000110001 |
| 0110010101 | 000110010 |
| 0111011100 | 000110011 |
| 1000100011 | 000110100 |
| 1001101010 | 000110101 |
| 1010110001 | 000110110 |
| 1010111000 | 000110111 |
| 1001111000 | 000111000 |
| 1000110001 | 000111001 |
| 0111101010 | 000111010 |
| 0110100011 | 000111011 |
| 0101011100 | 000111100 |
| 0100010101 | 000111101 |
| 0011001110 | 000111110 |
| 0010000110 | 000111111 |
| 0001000101 | 001000000 |
| 0000001100 | 001000001 |
| 0000010011 | 001000010 |
| 0001011010 | 001000011 |
| 0010100001 | 001000100 |
| 0011101000 | 001000101 |
| 0100110000 | 001000110 |
| 0101111001 | 001000111 |
| 0110111010 | 001001000 |
| 0111110011 | 001001001 |
| 1000101100 | 001001010 |
| 1001100101 | 001001011 |

| | |
|---|---|
| 1010011110 | 001001100 |
| 1010010110 | 001001101 |
| 1001001101 | 001001110 |
| 1000000100 | 001001111 |
| 0111000011 | 001010000 |
| 0110001010 | 001010001 |
| 0101010001 | 001010010 |
| 0100011000 | 001010011 |
| 0011100000 | 001010100 |
| 0010101001 | 001010101 |
| 0001110010 | 001010110 |
| 0000111011 | 001010111 |
| 0000111100 | 001011000 |
| 0001110101 | 001011001 |
| 0010101110 | 001011010 |
| 0011100110 | 001011011 |
| 0100011101 | 001011100 |
| 0101010100 | 001011101 |
| 0110001011 | 001011110 |
| 0111000010 | 001011111 |
| 1000000001 | 001100000 |
| 1001001000 | 001100001 |
| 1010010000 | 001100010 |
| 1010011001 | 001100011 |
| 1001100010 | 001100100 |
| 1000101011 | 001100101 |
| 0111110100 | 001100110 |
| 0110111101 | 001100111 |
| 0101111110 | 001101000 |
| 0100110110 | 001101001 |
| 0011101101 | 001101010 |
| 0010100100 | 001101011 |
| 0001011011 | 001101100 |
| 0000010010 | 001101101 |
| 0000001001 | 001101110 |
| 0001000000 | 001101111 |
| 0010000000 | 001110000 |
| 0011001001 | 001110001 |
| 0100010010 | 001110010 |
| 0101011011 | 001110011 |
| 0110100100 | 001110100 |
| 0111101101 | 001110101 |
| 1000110110 | 001110110 |
| 1001111110 | 001110111 |
| 1010111101 | 001111000 |

| | |
|---|---|
| 1010110100 | 001111001 |
| 1001101011 | 001111010 |
| 1000100010 | 001111011 |
| 0111011001 | 001111100 |
| 0110010000 | 001111101 |
| 0101001000 | 001111110 |
| 0100000001 | 001111111 |
| 0011000010 | 010000000 |
| 0010001011 | 010000001 |
| 0001010100 | 010000010 |
| 0000011101 | 010000011 |
| 0000100110 | 010000100 |
| 0001101110 | 010000101 |
| 0010110101 | 010000110 |
| 0011111100 | 010000111 |
| 0100111011 | 010001000 |
| 0101110010 | 010001001 |
| 0110101001 | 010001010 |
| 0111100000 | 010001011 |
| 1000011000 | 010001100 |
| 1001010001 | 010001101 |
| 1010001010 | 010001110 |
| 1010000011 | 010001111 |
| 1001000100 | 010010000 |
| 1000001101 | 010010001 |
| 0111010110 | 010010010 |
| 0110011110 | 010010011 |
| 0101100101 | 010010100 |
| 0100101100 | 010010101 |
| 0011110011 | 010010110 |
| 0010111010 | 010010111 |
| 0001111001 | 010011000 |
| 0000110000 | 010011001 |
| 0000101000 | 010011010 |
| 0001100001 | 010011011 |
| 0010011010 | 010011100 |
| 0011010011 | 010011101 |
| 0100001100 | 010011110 |
| 0101000101 | 010011111 |
| 0110000110 | 010100000 |
| 0111001110 | 010100001 |
| 1000010101 | 010100010 |
| 1001011100 | 010100011 |
| 1010100011 | 010100100 |
| 1010101010 | 010100101 |

| | |
|---|---|
| 1001110001 | 010100110 |
| 1000111000 | 010100111 |
| 0111111000 | 010101000 |
| 0110110001 | 010101001 |
| 0101101010 | 010101010 |
| 0100100011 | 010101011 |
| 0011011100 | 010101100 |
| 0010010101 | 010101101 |
| 0001001110 | 010101110 |
| 0000000110 | 010101111 |
| 0000000101 | 010110000 |
| 0001001100 | 010110001 |
| 0010010011 | 010110010 |
| 0011011010 | 010110011 |
| 0100100001 | 010110100 |
| 0101101000 | 010110101 |
| 0110110000 | 010110110 |
| 0111111001 | 010110111 |
| 1000111010 | 010111000 |
| 1001110011 | 010111001 |
| 1010101100 | 010111010 |
| 1010100101 | 010111011 |
| 1001011110 | 010111100 |
| 1000010110 | 010111101 |
| 0111001101 | 010111110 |
| 0110000100 | 010111111 |
| 0101000011 | 011000000 |
| 0100001010 | 011000001 |
| 0011010001 | 011000010 |
| 0010011000 | 011000011 |
| 0001100000 | 011000100 |
| 0000101001 | 011000101 |
| 0000110010 | 011000110 |
| 0001111011 | 011000111 |
| 0010111100 | 011001000 |
| 0011110101 | 011001001 |
| 0100101110 | 011001010 |
| 0101100110 | 011001011 |
| 0110011101 | 011001100 |
| 0111010100 | 011001101 |
| 1000001011 | 011001110 |
| 1001000010 | 011001111 |
| 1010000001 | 011010000 |
| 1010001000 | 011010001 |
| 1001010000 | 011010010 |

```
1000011001 011010011
0111100010 011010100
0110101011 011010101
0101110100 011010110
0100111101 011010111
0011111110 011011000
0010110110 011011001
0001101101 011011010
0000100100 011011011
0000011011 011011100
0001010010 011011101
0010001001 011011110
0011000000 011011111
0100000000 011100000
0101001001 011100001
0110010010 011100010
0111011011 011100011
1000100100 011100100
1001101101 011100101
1010110110 011100110
1010111110 011100111
1001111101 011101000
1000110100 011101001
0111101011 011101010
0110100010 011101011
0101011001 011101100
0100010000 011101101
0011001000 011101110
0010000001 011101111
0001000010 011110000
0000001011 011110001
0000010100 011110010
0001011101 011110011
0010100110 011110100
0011101110 011110101
0100110101 011110110
0101111100 011110111
0110111011 011111000
0111110010 011111001
1000101001 011111010
1001100000 011111011
1010011000 011111100
1010010001 011111101
1001001010 011111110
1000000011 011111111
```

| | |
|---|---|
| 0111000100 | 100000000 |
| 0110001101 | 100000001 |
| 0101010110 | 100000010 |
| 0100011110 | 100000011 |
| 0011100101 | 100000100 |
| 0010101100 | 100000101 |
| 0001110011 | 100000110 |
| 0000111010 | 100000111 |
| 0000111001 | 100001000 |
| 0001110000 | 100001001 |
| 0010101000 | 100001010 |
| 0011100001 | 100001011 |
| 0100011010 | 100001100 |
| 0101010011 | 100001101 |
| 0110001100 | 100001110 |
| 0111000101 | 100001111 |
| 1000000110 | 100010000 |
| 1001001110 | 100010001 |
| 1010010101 | 100010010 |
| 1010011100 | 100010011 |
| 1001100011 | 100010100 |
| 1000101010 | 100010101 |
| 0111110001 | 100010110 |
| 0110111000 | 100010111 |
| 0101111000 | 100011000 |
| 0100110001 | 100011001 |
| 0011101010 | 100011010 |
| 0010100011 | 100011011 |
| 0001011100 | 100011100 |
| 0000010101 | 100011101 |
| 0000001110 | 100011110 |
| 0001000110 | 100011111 |
| 0010000101 | 100100000 |
| 0011001100 | 100100001 |
| 0100010011 | 100100010 |
| 0101011010 | 100100011 |
| 0110100001 | 100100100 |
| 0111101000 | 100100101 |
| 1000110000 | 100100110 |
| 1001111001 | 100100111 |
| 1010111010 | 100101000 |
| 1010110011 | 100101001 |
| 1001101100 | 100101010 |
| 1000100101 | 100101011 |
| 0111011110 | 100101100 |

| | |
|---|---|
| 0110010110 | 100101101 |
| 0101001101 | 100101110 |
| 0100000100 | 100101111 |
| 0011000011 | 100110000 |
| 0010001010 | 100110001 |
| 0001010001 | 100110010 |
| 0000011000 | 100110011 |
| 0000100000 | 100110100 |
| 0001101001 | 100110101 |
| 0010110010 | 100110110 |
| 0011111011 | 100110111 |
| 0100111100 | 100111000 |
| 0101110101 | 100111001 |
| 0110101110 | 100111010 |
| 0111100110 | 100111011 |
| 1000011101 | 100111100 |
| 1001010100 | 100111101 |
| 1010001011 | 100111110 |
| 1010000010 | 100111111 |
| 1001000001 | 101000000 |
| 1000001000 | 101000001 |
| 0111010000 | 101000010 |
| 0110011001 | 101000011 |
| 0101100010 | 101000100 |
| 0100101011 | 101000101 |
| 0011110100 | 101000110 |
| 0010111101 | 101000111 |
| 0001111110 | 101001000 |
| 0000110110 | 101001001 |
| 0000101101 | 101001010 |
| 0001100100 | 101001011 |
| 0010011011 | 101001100 |
| 0011010010 | 101001101 |
| 0100001001 | 101001110 |
| 0101000000 | 101001111 |
| 0110000000 | 101010000 |
| 0111001001 | 101010001 |
| 1000010010 | 101010010 |
| 1001011011 | 101010011 |
| 1010100100 | 101010100 |
| 1010101101 | 101010101 |
| 1001110110 | 101010110 |
| 1000111110 | 101010111 |
| 0111111101 | 101011000 |
| 0110110100 | 101011001 |

| | |
|---|---|
| 0101101011 | 101011010 |
| 0100100010 | 101011011 |
| 0011011001 | 101011100 |
| 0010010000 | 101011101 |
| 0001001000 | 101011110 |
| 0000000001 | 101011111 |
| 0000000010 | 101100000 |
| 0001001011 | 101100001 |
| 0010010100 | 101100010 |
| 0011011101 | 101100011 |
| 0100100110 | 101100100 |
| 0101101110 | 101100101 |
| 0110110101 | 101100110 |
| 0111111100 | 101100111 |
| 1000111011 | 101101000 |
| 1001110010 | 101101001 |
| 1010101001 | 101101010 |
| 1010100000 | 101101011 |
| 1001011000 | 101101100 |
| 1000010001 | 101101101 |
| 0111001010 | 101101110 |
| 0110000011 | 101101111 |
| 0101000100 | 101110000 |
| 0100001101 | 101110001 |
| 0011010110 | 101110010 |
| 0010011110 | 101110011 |
| 0001100101 | 101110100 |
| 0000101100 | 101110101 |
| 0000110011 | 101110110 |
| 0001111010 | 101110111 |
| 0010111001 | 101111000 |
| 0011110000 | 101111001 |
| 0100101000 | 101111010 |
| 0101100001 | 101111011 |
| 0110011010 | 101111100 |
| 0111010011 | 101111101 |
| 1000001100 | 101111110 |
| 1001000101 | 101111111 |
| 1010000110 | 110000000 |
| 1010001110 | 110000001 |
| 1001010101 | 110000010 |
| 1000011100 | 110000011 |
| 0111100011 | 110000100 |
| 0110101010 | 110000101 |
| 0101110001 | 110000110 |

| | |
|---|---|
| 0100111000 | 110000111 |
| 0011111000 | 110001000 |
| 0010110001 | 110001001 |
| 0001101010 | 110001010 |
| 0000100011 | 110001011 |
| 0000011100 | 110001100 |
| 0001010101 | 110001101 |
| 0010001110 | 110001110 |
| 0011000110 | 110001111 |
| 0100000101 | 110010000 |
| 0101001100 | 110010001 |
| 0110010011 | 110010010 |
| 0111011010 | 110010011 |
| 1000100001 | 110010100 |
| 1001101000 | 110010101 |
| 1010110000 | 110010110 |
| 1010111001 | 110010111 |
| 1001111010 | 110011000 |
| 1000110011 | 110011001 |
| 0111101100 | 110011010 |
| 0110100101 | 110011011 |
| 0101011110 | 110011100 |
| 0100010110 | 110011101 |
| 0011001101 | 110011110 |
| 0010000100 | 110011111 |
| 0001000011 | 110100000 |
| 0000001010 | 110100001 |
| 0000010001 | 110100010 |
| 0001011000 | 110100011 |
| 0010100000 | 110100100 |
| 0011101001 | 110100101 |
| 0100110010 | 110100110 |
| 0101111011 | 110100111 |
| 0110111100 | 110101000 |
| 0111110101 | 110101001 |
| 1000101110 | 110101010 |
| 1001100110 | 110101011 |
| 1010011101 | 110101100 |
| 1010010100 | 110101101 |
| 1001001011 | 110101110 |
| 1000000010 | 110101111 |
| 0111000001 | 110110000 |
| 0110001000 | 110110001 |
| 0101010000 | 110110010 |
| 0100011001 | 110110011 |

| | |
|---|---|
| 0011100010 | 110110100 |
| 0010101011 | 110110101 |
| 0001110100 | 110110110 |
| 0000111101 | 110110111 |
| 0000111110 | 110111000 |
| 0001110110 | 110111001 |
| 0010101101 | 110111010 |
| 0011100100 | 110111011 |
| 0100011011 | 110111100 |
| 0101010010 | 110111101 |
| 0110001001 | 110111110 |
| 0111000000 | 110111111 |
| 1000000000 | 111000000 |
| 1001001001 | 111000001 |
| 1010010010 | 111000010 |
| 1010011011 | 111000011 |
| 1001100100 | 111000100 |
| 1000101101 | 111000101 |
| 0111110110 | 111000110 |
| 0110111110 | 111000111 |
| 0101111101 | 111001000 |
| 0100110100 | 111001001 |
| 0011101011 | 111001010 |
| 0010100010 | 111001011 |
| 0001011001 | 111001100 |
| 0000010000 | 111001101 |
| 0000001000 | 111001110 |
| 0001000001 | 111001111 |
| 0010000010 | 111010000 |
| 0011001011 | 111010001 |
| 0100010100 | 111010010 |
| 0101011101 | 111010011 |
| 0110100110 | 111010100 |
| 0111101110 | 111010101 |
| 1000110101 | 111010110 |
| 1001111100 | 111010111 |
| 1010111011 | 111011000 |
| 1010110010 | 111011001 |
| 1001101001 | 111011010 |
| 1000100000 | 111011011 |
| 0111011000 | 111011100 |
| 0110010001 | 111011101 |
| 0101001010 | 111011110 |
| 0100000011 | 111011111 |
| 0011000100 | 111100000 |

```
0010001101 111100001
0001010110 111100010
0000011110 111100011
0000100101 111100100
0001101100 111100101
0010110011 111100110
0011111010 111100111
0100111001 111101000
0101110000 111101001
0110101000 111101010
0111100001 111101011
1000011010 111101100
1001010011 111101101
1010001100 111101110
1010000101 111101111
1001000110 111110000
1000001110 111110001
0111010101 111110010
0110011100 111110011
0101100011 111110100
0100101010 111110101
0011110001 111110110
0010111000 111110111
0001111000 111111000
0000110001 111111001
0000101010 111111010
0001100011 111111011
0010011100 111111100
0011010101 111111101
0100001110 111111110
0101000110 111111111
.e
```

# APPENDIX F: SPICE FILE FOR COMPARATOR GROUP 8

## A. Extracted Spice File

```
** SPICE file created for circuit compgrp8
**
** NODE: 0 = GND
** NODE: 1 = Vdd
** NODE: 2 = Error
M0 1 100 100 1 pfet L=3.0U W=12.0U
M1 101 100 1 1 pfet L=3.0U W=12.0U
M2 1 101 102 1 pfet L=2.0U W=4.0U
M3 103 102 1 1 pfet L=2.0U W=3.0U
M4 104 105 103 1 pfet L=2.0U W=3.0U
M5 106 107 104 1 pfet L=2.0U W=3.0U
M6 1 108 106 1 pfet L=2.0U W=3.0U
M7 108 104 1 1 pfet L=2.0U W=3.0U
M8 100 109 110 0 nfet L=10.0U W=10.0U
M9 101 111 110 0 nfet L=10.0U W=10.0U
 .                                      .
 .                                      .
 .                                      .
M113 173 172 0 0 nfet L=2.0U W=4.0U
M114 180 173 0 0 nfet L=2.0U W=3.0U
M115 175 107 180 0 nfet L=2.0U W=3.0U
M116 181 105 175 0 nfet L=2.0U W=3.0U
M117 0 177 181 0 nfet L=2.0U W=3.0U
M118 177 175 0 0 nfet L=2.0U W=3.0U
C0 181 0 18F
** NODE: 181 = 8_294_213#
** NODE: 180 = 8_260_213#
C1 178 0 62F
** NODE: 178 = 8_42_195#
C2 179 0 12F
** NODE: 179 = Vt1
** NODE: 176 = 8_306_157#
 .
 .
 .
C60 100 0 71F
** NODE: 100 = 8_42_884#
C61 1 0 674F
** NODE: 1 = Vdd!
```

## B. Spice File Ready for Simulation

```
 CMOS COMPARATOR GROUP 8 FROM MAGIC

** SPICE file created for circuit compgrp8
**
* SPICE3C PMOS and NMOS model level 2 parameters for the
* ORBIT Technology (SCNA20) 2.0U CMOS N-well process.
* DATED: 20 DEC 94

** Transistor Model Parameters for ORBIT Technology.
.MODEL nnf NMOS(LEVEL=2 PHI=0.600 TOX=4.3500E-08 XJ=0.200U TPG=1
+ VTO=0.8756 DELTA=8.5650E+00 LD=2.3950E-07 KP=4.5494E-05
+ UO=573.1 UEXP=1.5920E-01 UCRIT=5.9160E+04 RSH=1.0310E+01
+ GAMMA=0.4179 NSUB=3.3160E+15 NFS=8.1800E+12 VMAX=6.0280E+04
+ CGB0=4.0921E-10 CJ=1.0375E-04 MJ=0.6604 CJSW=2.1694E-10
+ MJSW=0.178543 PB=0.800
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -4.0460E-07

.MODEL npf PMOS(LEVEL=2 PHI=0.600 TOX=4.3500E-08 XJ=0.200U TPG=-1
+ VTO=-0.8889 DELTA=4.8720E+00 LD=2.9230E-07 KP=1.5035E-05
+ UO=189.4 UEXP=2.7910E-01 UCRIT=9.5670E+04 RSH=1.8180E+01
+ GAMMA=0.7327 NSUB=1.0190E+16 NFS=6.1500E+12 VMAX=9.9990E+05
+ CGB0=4.0605E-10 CJ=3.2456E-04 MJ=0.6044 CJSW=2.5430E-10
+ MJSW=0.244194 PB=0.800
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -3.6560E-07

*Sources
Vdd 1  0 5V
Vss 113 0 -5V
Vb  112 0 -1.6V

*Input signal
*Vin  109 0
Vin  109 0 PWL 0s 2.8194V, 4.522ms 636.1060mV, 9.044ms 2.8194, R

*Clocks for Latch Circuit (10KHz)
VCLK2 107  0  pulse(0v 5v 0.0s 1ns 1ns  .05ms .1ms)
VCLK1 105  0  pulse(0v 5V .05ms 1ns 1ns .05ms .1ms)

*Comparator Threshold Voltages
Vt1 179 0  2.6376V
Vt2 168 0  2.3601V
```

```
Vt3 157 0  2.0528V
Vt4 146 0  1.7187V
Vt5 135 0  1.3676V
Vt6 124 0  1.0554V
Vt7 111 0  0.7763925V

** NODE: 0 = GND
** NODE: 1 = Vdd
** NODE: 2 = Error
M0 1 100 100 1 npf L=3.0U W=12.0U
M1 101 100 1 1 npf L=3.0U W=12.0U
M2 1 101 102 1 npf L=2.0U W=4.0U
M3 103 102 1 1 npf L=2.0U W=3.0U
M4 104 105 103 1 npf L=2.0U W=3.0U
M5 106 107 104 1 npf L=2.0U W=3.0U
M6 1 108 106 1 npf L=2.0U W=3.0U
M7 108 104 1 1 npf L=2.0U W=3.0U
M8 100 109 110 113 nnf L=10.0U W=10.0U
M9 101 111 110 113 nnf L=10.0U W=10.0U
 .                                        .
 .                                        .
 .                                        .
M113 173 172 0 113 nnf L=2.0U W=4.0U
M114 180 173 0 113 nnf L=2.0U W=3.0U
M115 175 107 180 113 nnf L=2.0U W=3.0U
M116 181 105 175 113 nnf L=2.0U W=3.0U
M117 0 177 181 113 nnf L=2.0U W=3.0U
M118 177 175 0 113 nnf L=2.0U W=3.0U
C0 181 0 18F
** NODE: 181 = 8_294_213#
** NODE: 180 = 8_260_213#
C1 178 0 62F
** NODE: 178 = 8_42_195#
C2 179 0 12F
** NODE: 179 = Vt1
** NODE: 176 = 8_306_157#
C3 177 0 46F
** NODE: 177 = X1
C4 175 0 56F
** NODE: 175 = 8_272_213#
** NODE: 174 = 8_260_157#
C5 173 0 44F
** NODE: 173 = 8_126_205#
C6 172 0 72F
** NODE: 172 = 8_74_161#
```

```
C7 171 0 71F
** NODE: 171 = 8_42_161#
C8 170 0 18F
** NODE: 170 = 8_294_39#
** NODE: 169 = 8_260_39#
          .
          .
          .
C58 105 0 189F
** NODE: 105 = -CLK
C59 101 0 72F
** NODE: 101 = 8_74_884#
C60 100 0 71F
** NODE: 100 = 8_42_884#
C61 1 0 674F
** NODE: 1 = Vdd!


****Simulation Parameters********

.option dcon=1 post probe

.tran 19us 28ms
*.dc Vin  0.0  15.0  1mv

.probe tran V(109) V(105) V(107) V(108) V(122) V(133) V(144) V(155)
+ V(166) V(177) I(1)

*.probe dc V(109) V(105) V(107) V(108) V(122) V(133) V(144) V(155)
+ V(166) V(177) I(1)

*.print tran V(109) V(108) V(122) V(133) V(144) V(155) V(166) V(177) I(1)
*.print dc V(109) V(108) V(122) V(133) V(144) V(155) V(166) V(177) I(1)

.end
```

# APPENDIX G: MODULUS 11 DATA AND MATLAB M-FILE CODE

## A. Modulus 11 Data

```
% cmos mod 11 folding/comparator/latch ckt from magic for 9-bit adc

%              Vin    Fold     C1      C2      C3      C4      C5
%      time    volt   volt     volt    volt    volt    volt    volt
%              103    280      279     345     356     367     378

Vmod11a=[0    0   2.8011     2.4116e-6 2.4116e-6 2.4116e-6 2.4116e-6 2.4116e-6 ;
 19.00000e-6  988.0000e-6  2.8010  2.8047e-6 2.7107e-6 2.7107e-6 2.7107e-6 2.7107e-6 ;
 38.00000e-6  1.9760e-3  2.8006   2.4528e-6 2.2196e-6 2.2196e-6 2.2196e-6 2.2196e-6 ;
 57.00000e-6  2.9640e-3  2.8002   2.3994e-6 2.3994e-6 2.3994e-6 2.3994e-6 2.3994e-6 ;
      .              .          .           .            .            .           .
      .
      .              .          .           .            .            .           .
      .
      .              .          .           .            .            .           .
      .
 99.86400e-3  5.1929   2.3997   2.4050e-6  2.4050e-6  2.4050e-6  2.4050e-6  2.4050e-6 ;
 99.88300e-3  5.1939   2.3927   2.4181e-6  2.4181e-6  2.4181e-6  2.4181e-6  2.4181e-6 ;
 99.90200e-3  5.1949   2.3857   1.3135e-6  1.3695e-6  1.3695e-6  1.3695e-6  1.3697e-6 ;
 99.92100e-3  5.1959  2.3787  142.4592e-9  74.0352e-9  74.0360e-9  74.1926e-6  74.9447e-9 ;
 99.94000e-3  5.1969   2.3716   2.4117e-6  2.4117e-6  2.4117e-6  2.4117e-6  2.4117e-6 ;
 99.95900e-3  5.1979   2.3645   2.4263e-6  2.4263e-6  2.4263e-6  2.4263e-6  2.4263e-6 ;
 99.97800e-3  5.1989   2.3574   2.3995e-6  2.3995e-6  2.3995e-6  2.3995e-6  2.3995e-6 ;
 99.99700e-3  5.1998   2.3503   2.3872e-6  2.3872e-6  2.3872e-6  2.3872e-6  2.3872e-6 ];

%              Vin     C6      C7      C8      C9      C10
%      time    volt    volt    volt    volt    volt    volt
%              103     389     400     411     492     632

Vmod11b=[0  0  2.4116e-6  2.4116e-6  2.4116e-6  2.4116e-6  2.4116e-6 ;
 19.00000e-6    988.0000e-6  2.7106e-6  2.7103e-6  2.7099e-6  2.7092e-6  2.7859e-6 ;
 38.00000e-6  1.9760e-3  2.2197e-6  2.2199e-6  2.2201e-6  2.2206e-6  2.4364e-6 ;
 57.00000e-6  2.9640e-3  2.3994e-6  2.3994e-6  2.3994e-6  2.3994e-6  2.3993e-6 ;
      .             .        .            .            .            .
      .             .        .            .            .            .
      .             .        .            .            .            .
 99.86400e-3  5.1929   2.4050e-6  2.4050e-6  2.4050e-6  5.0000   5.0000  ;
 99.88300e-3  5.1939   2.4181e-6  2.4182e-6  2.4182e-6  5.0000   5.0000  ;
99.90200e-3  5.1949   1.3701e-6  1.3707e-6  1.3721e-6  5.0000   5.0000  ;
99.92100e-3  5.1959  76.3168e-9    78.6180e-9    84.1740e-9  5.0000   5.0000  ;
99.94000e-3  5.1969   2.4117e-6  2.4117e-6  2.4117e-6  5.0000   5.0000  ;
99.95900e-3  5.1979   2.4264e-6  2.4264e-6  2.4265e-6  5.0000   5.0000  ;
99.97800e-3  5.1989   2.3995e-6  2.3994e-6  2.3993e-6  5.0000   5.0000  ;
99.99700e-3  5.1998   2.3872e-6  2.3871e-6  2.3870e-6  5.0000   5.0000];
```

## B. MATLAB m-File Code to Plot Waveforms

```
% m11plot.m
% This M-File will call the Modulus 11 Comparator Output (data)
% matrices and plot them against the input sigal

m11data
orient tall

subplot(611),plot(Vmod11a(:,2),Vmod11a(:,3))
title('MOD 11')
ylabel('Folding Output')
axis([0 5 -2 6])

subplot(612),plot(Vmod11a(:,2),Vmod11a(:,4))
ylabel('C1')
axis([0 5 -2 6])

subplot(613),plot(Vmod11a(:,2),Vmod11a(:,5))
ylabel('C2')
axis([0 5 -2 6])

subplot(614),plot(Vmod11a(:,2),Vmod11a(:,6))
ylabel('C3')
axis([0 5 -2 6])

subplot(615),plot(Vmod11a(:,2),Vmod11a(:,7))
ylabel('C4')
axis([0 5 -2 6])

subplot(616),plot(Vmod11a(:,2),Vmod11b(:,2))
ylabel('C5')
axis([0 5 -2 6])
xlabel('ADC Input Voltage')
print -Ppr12 fig1
```

```
orient tall

subplot(611),plot(Vmod11a(:,2),Vmod11a(:,3))
title('MOD 11')
ylabel('Folding Output')
axis([0 5 -2 6])

subplot(612),plot(Vmod11a(:,2),Vmod11b(:,3))
ylabel('C6')
axis([0 5 -2 6])

subplot(613),plot(Vmod11a(:,2),Vmod11b(:,4))
ylabel('C7')
axis([0 5 -2 6])

subplot(614),plot(Vmod11a(:,2),Vmod11b(:,5))
ylabel('C8')
axis([0 5 -2 6])

subplot(615),plot(Vmod11a(:,2),Vmod11b(:,6))
ylabel('C9')
axis([0 5 -2 6])

subplot(616),plot(Vmod11a(:,2),Vmod11b(:,7))
ylabel('C10')
xlabel('ADC Input Voltage')
axis([0 5 -2 6])
print -Ppr12 fig2
```

# APPENDIX H: ESIM TEST CODE AND RESULTS FOR 2-INPUT XOR GATE, INVERTER AND EVEN-PARITY CIRCUIT

** Test code for testing 2-Input XOR Gate, Inverter Gate and Even-Parity Circuit.
** Read in column format.

## A. Test Code and Results for 2-Input XOR Gate

```
% esim xor2.sim
ESIM (V3.5 03/27/91)
8 transistors, 7 nodes (0 pulled up)
sim> w IN1 IN2 OUT
sim> V IN1 0011
sim> V IN2 0101
sim> I
initialization took 6 steps
sim> I
initialization took 0 steps
sim> G
>0011:IN1
>0101:IN2
>0110:OUT
sim> quit
```

## B. Test Code and Results for Inverter Gate

```
% esim inverter.sim
ESIM (V3.5 03/27/91)
2 transistors, 4 nodes (0 pulled up)
sim> w IN OUT
sim> V IN 01
sim> I
initialization took 2 steps
sim> I
initialization took 0 steps
sim> G
>01:IN
>10:OUT
sim> quit
```

## C. Test Code and Results for Even-Parity Circuit

```
% esim parity.sim
ESIM (V3.5 03/27/91)
106 transistors, 56 nodes (0 pulled up)
sim> w V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 Y1
sim> V V1 000000000000001
sim> V V2 000000000000011
sim> V V3 000000000000111
sim> V V4 000000000001111
sim> V V5 000000000011111
sim> V V6 000000000111111
sim> V V7 000000001111111
sim> V V8 000000011111111
sim> V V9 000000111111111
sim> V V10 000001111111111
sim> V V11 000011111111111
sim> V V12 000111111111111
sim> V V13 001111111111111
sim> V V14 011111111111111
sim> I
initialization took 100 steps
sim> I
initialization took 0 steps
sim> G
>000000000000001:V1
>000000000000011:V2
>000000000000111:V3
>000000000001111:V4
>000000000011111:V5
>000000000111111:V6
>000000001111111:V7
>000000011111111:V8
>000000111111111:V9
>000001111111111:V10
>000011111111111:V11
>000111111111111:V12
>001111111111111:V13
>011111111111111:V14
>101010101010101:Y1
sim> quit
```

# APPENDIX I: ESIM TEST CODE AND RESULTS FOR MODULUS 7, 8 AND 11 PLAs

** Test Code for testing Modulus PLAs.
** INPUTS: X1-X*xx*, OUTPUTS: Y1-Y*x*.
** Read in column format.

## A. Test Code and Results for Modulus 7 PLA.

```
%esim mod7pla.sim
ESIM (V3.5 03/27/91)
81 transistors, 33 nodes, (10 pulled up)
sim> w X6 X5 X4 X3 X2 X1 Y3 Y2 Y1
sim> V X6 0000001
sim> V X5 0000011
sim> V X4 0000111
sim> V X3 0001111
sim> V X2 0011111
sim> V X1 0111111
sim> I
initialization took 50 steps
sim> I
initialization took 0 steps
sim> G
>0000001:X6
>0000011:X5
>0000111:X4
>0001111:X3
>0011111:X2
>0111111:X1

>0000111:Y3
>0011001:Y2
>0101010:Y1

sim> quit
```

## B. Test Code and Results for Modulus 8 PLA.

```
%esim mod8pla.sim
ESIM (V3.5 03/27/91)
102 transistors, 37 nodes, (11 pulled up)
sim> w X7 X6 X5 X4 X3 X2 X1 Y3 Y2 Y1
sim> V X7 00000001
sim> V X6 00000011
sim> V X5 00000111
sim> V X4 00001111
sim> V X3 00011111
sim> V X2 00111111
sim> V X1 01111111
sim> I
initialization took 54 steps
sim> I
initialization took 0 steps
sim> G
>00000001:X7
>00000011:X6
>00000111:X5
>00001111:X4
>00011111:X3
>00111111:X2
>01111111:X1

>00001111:Y3
>00110011:Y2
>01010101:Y1

sim> quit
```

## C. Test Code and Results for Modulus 11 PLA.

```
%esim mod11pla.sim
ESIM (V3.5 03/27/91)
175 transistors, 51 nodes, (15 pulled up)
sim> w X10 X9 X8 X7 X6 X5 X4 X3 X2 X1 Y4 Y3 Y2 Y1
sim> V X10 00000000001
sim> V X9 00000000011
sim> V X8 00000000111
sim> V X7 00000001111
sim> V X6 00000011111
sim> V X5 00000111111
sim> V X4 00001111111
sim> V X3 00011111111
sim> V X2 00111111111
sim> V X1 01111111111
sim> I
initialization took 78 steps
sim> I
initialization took 0 steps
sim> G
>00000000001:X10
>00000000011:X9
>00000000111:X8
>00000001111:X7
>00000011111:X6
>00000111111:X5
>00001111111:X4
>00011111111:X3
>00111111111:X2
>01111111111:X1

>00000000111:Y4
>00001111000:Y3
>00110011001:Y2
>01010101010:Y1

sim> quit
```

# APPENDIX J: ESIM TEST CODE AND RESULTS FOR FINAL PLA

```
** Test package of all 616 possible vectors plus a few extra to test Large
** PLA and Error checking circuit (D10).
** INPUTS: Y1 --> Y10, OUTPUTS: D1 --> D9, ERROR CHECKING: d10
** Read by column of inputs (Y1-Y10) and match to column of outputs (D1-D9).
** Mod 7 outputs = Y1-Y3, Mod8 outputs = Y4-Y6, Mod 11 outputs = Y7-Y10.
** For example, first column Y1-Y10 are all set to 0's, therefore output of Large
** PLA should be 0's (check first column of D1-D9). The second column indicates
** a 1 out of each modulus PLA, therefore Large PLA should be 1 (check second
** column of D1-D9). D10 will always be 0 unless the input is not one of 616 possible
** input code from the Modulus PLAs. The last part of the test is dedicated to this
** error checking.

>0000000011111100000000000000000111111:Y10

>00001111000000011110000000001111000000:Y9

>001100110011001100110000110011001100:Y8

>01010101010010101010100101010101010010:Y7

>000011111111100000000011111111100000000:Y6

>001100111100110000110011110011000011:Y5

>010101011010101001010101101010100101:Y4

>00001111111000000001111111000000001111:Y3

>001100110011000011001100110000110011:Y2

>010101001010100101010010101001010100:Y1


>000000000000000000000000000000000000:D10

>000000000000000000000000000000000000:D9

>000000000000000000000000000000000000:D8

>000000000000000000000000000000000000:D7

>000000000000000000000000000000001111:D6
```

```
>000000000000000011111111111111110000:D5

>000000001111111100000000111111110000:D4

>000011110000111100001111000011110000:D3

>001100110011001100110011001100110011:D2

>010101010101010101010101010101010101:D1

>0000000000000000111111000000000000000000111111000000:Y10

>111100000000011110000000111100000000011110000000111000:Y9

>110011000011001100110011001100001100110011001100110:Y8

>1010101001010101010100101010101010010101010100101010101:Y7

>1111111100000000111111110000000011111111000000000111:Y6

>001111001100001100111100110000110011110011000011001:Y5

>010110101010010101011010101001010101101010100101010:Y4

>110000000011111100000000111111000000000111111000000:Y3

>001100001100110011000011001100110000110011001100001:Y2

>101010010101001010100101010010101001010100101010010:Y1


>000000000000000000000000000000000000000000000000:D10

>000000000000000000000000000000000000000000000000:D9

>000000000000000000000000000000000000000000000000:D8

>000000000000000000000000000111111111111111111111:D7

>111111111111111111111111111000000000000000000000:D6

>000000000000111111111111111100000000000000001111111:D5

>000011111111000000000111111110000000001111111110000000:D4
```

```
>111100001111000011110000111100001111000011110000111:D3

>001100110011001100110011001100110011001100110011001:D2

>010101010101010101010101010101010101010101010101010:D1


>000000000111111000000000000000011111100000000000000:Y10

>000001111000000111100000000111100000011110000000011:Y9

>000110011001100110011000011001100110011000011001100:Y8

>001010101010010101010100101010101001010101010010101:Y7

>111110000000011111111000000001111111100000000111111:Y6

>111001100001100111100110000110011110011000011001111:Y5

>110101010010101011010101001010101101010100101010110:Y4

>011111100000000111111000000001111110000000011111100:Y3

>100110011000011001100110000110011001100001100110011:Y2

>101001010100101010010101001010100101010010101001010:Y1


>000000000000000000000000000000000000000000000000000:D10

>000000000000000000000000000000000000000000000000000:D9

>000000000000000000000000000000000000000001111111111:D8

>111111111111111111111111111111111111111110000000000:D7

>000000000111111111111111111111111111111110000000000:D6

>111111111000000000000000011111111111111110000000000:D5

>011111111000000001111111100000000111111110000000011:D4

>100001111000011110000111100001111000011110000111100:D3

>100110011001100110011001100110011001100110011001100:D2
```

```
>101010101010101010101010101010101010101010101010101:D1

>00111111000000000000000011111100000000000000000011111:Y10

>110000001111000000000111100000001111000000000111100000:Y9

>110011001100110000110011001100110011000011001100110:Y8

>010100101010101001010101010010101010100101010101001:Y7

>11000000001111111100000000111111110000000011111110:Y6

>001100001100111100110000110011110011000011001111001:Y5

>101010010101011010101001010101101010100101010110101:Y4

>00000011111100000000011111100000000011111100000000111:Y3

>000011001100110000110011001100001100110011000011001:Y2

>100101010010101001010100101010010101001010100101010:Y1

>000000000000000000000000000000000000000000000000000:D10

>000000000000000000000000000000000000000000000000000:D9

>111111111111111111111111111111111111111111111111111:D8

>000000000000000000000000000000000000000000000000000:D7

>000000000000000000000011111111111111111111111111111:D6

>000000111111111111111100000000000000001111111111111:D5

>111111000000000111111111000000000111111110000000011111:D4

>001111000011110000111100001111000011110000111100001:D3

>110011001100110011001100110011001100110011001100110:D2

>010101010101010101010101010101010101010101010101010:D1
```

```
>100000000000000001111110000000000000000111111000000:Y10

>011110000000011110000001111000000001111000000111100:Y9

>011001100001100110011001100110000110011001100110011:Y8

>010101010010101010100101010101001010101010010101010:Y7

>000000011111111000000001111111100000000111111110000:Y6

>100001100111100110000110011110011000011001111001100:Y5

>010010101011010101001010101101010100101010110101010:Y4

>111000000001111110000000011111100000000111111000000:Y3

>100110000110011001100001100110011000011001100110000:Y2

>010101001010100101010010101001010100101010010101001:Y1

>000000000000000000000000000000000000000000000000000:D10

>000000000000000000000000000000000000000000000000000:D9

>111111111111111111111111111111111111111111111111111:D8

>000111111111111111111111111111111111111111111111111:D7

>111000000000000000000000000000000001111111111111111:D6

>111000000000000000011111111111111110000000000000000:D5

>111000000001111111100000000111111110000000011111111:D4

>111000011110000111100001111000011110000111100001111:D3

>011001100110011001100110011001100110011001100110011:D2

>101010101010101010101010101010101010101010101010101:D1

>000000000011111100000000000000001111110000000000000:Y10

>000000111100000011110000000011110000001111000000001:Y9
```

```
>000011001100110011001100001100110011001100110000110:Y8

>1001010101010010101010100101010101001010101010001010:Y7

>000011111111000000001111111100000000111111110000000:Y6

>001100111100110000110011110011000011001111001100001:Y5

>010101011010101001010101101010100101010110101010010:Y4

>001111110000000011111100000000111111000000001111110:Y3

>110011001100001100110011000011001100110000110011001:Y2

>010100101010010101001010100101010010101001010100101:Y1

>000000000000000000000000000000000000000000000000000:D10

>000000000000000111111111111111111111111111111111111:D9

>111111111111111100000000000000000000000000000000000:D8

>111111111111111100000000000000000000000000000000000:D7

>111111111111111100000000000000000000000000000000111:D6

>111111111111111100000000000000001111111111111111000:D5

>000000001111111100000000111111110000000011111111000:D4

>000011110000111100001111000011110000111100001111000:D3

>001100110011001100110011001100110011001100110011001:D2

>010101010101010101010101010101010101010101010101010:D1

>0001111110000000000000000111111000000000000000001111:Y10

>111000000111100000000011110000001111000000000111110000:Y9

>011001100110011000011001100110011001100001100110011:Y8

>101010010101010100101010101010010101010100101010100:Y7
```

```
>01111111100000000111111110000000001111111110000000011:Y6
>100111100110000110011110011000011001111001100001100:Y5
>101011010101001010101101010100101010110101010010101:Y4
>000000011111100000000011111100000000011111100000000011:Y3
>100001100110011000011001100110000110011001100001100:Y2
>010010101001010100101010010101001010100101010010101:Y1
>00000000000000000000000000000000000000000000000000:D10
>11111111111111111111111111111111111111111111111111:D9
>00000000000000000000000000000000000000000000000000:D8
>000000000000000000000000000001111111111111111111111:D7
>111111111111111111111111111110000000000000000000000:D6
>00000000000001111111111111110000000000000000111111:D5
>00000111111110000000001111111100000000011111111000000:D4
>01111000011110000111100001111000011110000111100011:D3
>100110011001100110011001100110011001100110011001100:D2
>101010101010101010101010101010101010101010101010101:D1

>110000000000000000111111000000000000000000111111100000:Y10
>0011110000000001111000000011110000000001111000000011110:Y9
>001100110000110011001100110011000011001100110011001:Y8
>101010101001010101010010101010100101010101001010101:Y7
>111110000000011111111000000000111111110000000011111:Y6
>111100110000110011110011000011001111001100001100111:Y5
```

```
>0110101010010101011010101001010101101010100101011:Y4

>111100000000111111000000001111110000000011111100000:Y3

>1100110000110011001100001100110011000011001100110 00:Y2

>0010101001010100101010010101001010100101010010101 00:Y1

>000000000000000000000000000000000000000000000000:D10

>111111111111111111111111111111111111111111111111:D9

>000000000000000000000000000000000000000011111111 1:D8

>111111111111111111111111111111111111111100000000 0:D7

>000000000111111111111111111111111111111100000000 0:D6

>111111111100000000000000011111111111111100000000 0:D5

>001111111100000000111111110000000011111111000000001:D4

>110000111100001111000011110000111100001111000011110:D3

>110011001100110011001100110011001100110011001100110:D2

>010101010101010101010101010101010101010101010101 0:D1

>000000000011111100000000000000000111111000000000000:Y10

>00000011110000001111000000000111100000001111000000000:Y9

>1000011001100110011001100001100110011001100110000 11:Y8

>0100101010101001010101010010101010100101010101001 01:Y7

>111000000000111111110000000001111111100000000011111111:Y6

>100110000110011110011000011001111001100001100111100:Y5

>010101001010101101010100101010110101010010101011010:Y4

>000111111000000001111110000000011111100000000111111:Y3
```

```
>011001100110000110011001100001100110011000011001100:Y2

>101010010101001010100101010010101001010100101010010:Y1


>000000000000000000000000000000000000000000000000:D10

>111111111111111111111111111111111111111111111111:D9

>111111111111111111111111111111111111111111111111:D8

>000000000000000000000000000000000000000000000000:D7

>000000000000000000000000111111111111111111111111:D6

>000000111111111111111111000000000000000000111111111111:D5

>111111000000000111111110000000011111111000000001111:D4

>000111100000111100000111100000111100000111100000111110000:D3

>011001100110011001100110011001100110011001100110011:D2

>101010101010101010101010101010101010101010101010101:D1


>000011111100000000000000000111111000000000000000000111:Y10

>111100000001111000000000111100000011110000000001111000:Y9

>001100110011001100001100110011001100110000110011001:Y8

>010101001010101010100101010101010010101010101001010101010:Y7

>000000001111111100000000111111110000000011111111000:Y6

>110000110011110011000011001111001100001100111100110:Y5

>101001010101101010100101010110101010010101011010101:Y4

>000000001111111000000000111111100000000011111110000000001:Y3

>110000110011001100001100110011000011001100110000110:Y2

>101001010100101010010101001010100101010010101001010:Y1
```

```
>000000000000000000000000000000000000000000000000000:D10

>111111111111111111111111111111111111111111111111111:D9

>111111111111111111111111111111111111111111111111111:D8

>000011111111111111111111111111111111111111111111111:D7

>111100000000000000000000000000000000111111111111111:D6

>111100000000000000001111111111111111000000000000000:D5

>111100000000111111110000000011111111000000001111111:D4

>111100001111000011110000111100001111000011110000111:D3

>001100110011001100110011001100110011001100110011001:D2

>010101010101010101010101010101010101010101010101010:D1


>11100000000000000:Y10

>00011110000000011:Y9

>10011001100001100:Y8

>01010101010010101:Y7

>00000111111110000:Y6

>00011001111001100:Y5

>00101010110101010:Y4

>11111000000001111:Y3

>01100110000110011:Y2

>10010101001010100:Y1


>00000000000000000:D10

>11111111111111111:D9
```

```
>11111111111111111:D8

>11111111111111111:D7

>11111111111111111:D6

>01111111111111111:D5

>10000000011111111:D4

>10000111100001111:D3

>10011001100110011:D2

>10101010101010101:D1
```

** Testing of Error Checking Circuit begins. Note the possible input vectors are not
** possible with the Modulus PLAs and that D10 is SET for each incorrect input.

```
>001111110000000000000000111111000000000000000000111111000000000000000:Y10

>110000000111000000000111000000011100000000011100000001110000000001:Y9

>110011001100110000110011001100110011000011001100110011001100001 10:Y8

>010100101010101001010101010010101010100101010101001010101010010 10:Y7

>000011111111000000000111111110000000001111111100000000111111100000:Y6

>001100111100110000110011110011000011001111001100001100111100110 00:Y5

>010101011010101001010101101010100101010110101010010101011010101 00:Y4

>110000000011111100000000011111100000000011111100000000011111100000000:Y3

>001100001100110011000011001100110000110011001100001100110011000 01:Y2

>101010010101001010100101010010101001010100101010010101001010100 10:Y1


>111111111111111111111111111111111111111111111111111111111111111111:D10

>000000000000000000000000000000000000000000000000000000000000000000:D9
```

```
>000000000000000000000000000000000000000000000000000000000000000000000000:D8
>000000000000000000000000000000000000000000000000000000000000000000000000:D7
>000000000000000000000000000000000000000000000000000000000000000000000000:D6
>000000000000000000000000000000000000000000000000000000000000000000000000:D5
>000000000000000000000000000000000000000000000000000000000000000000000000:D4
>000000000000000000000000000000000000000000000000000000000000000000000000:D3
>000000000000000000000000000000000000000000000000000000000000000000000000:D2
>000000000000000000000000000000000000000000000000000000000000000000000000:D1
>000111111000000000000000111111000000000:Y10
>111000000111100000000111100000011110000:Y9
>011001100110011000011001100110011001100:Y8
>101010010101010100101010101001010101010:Y7
>00011111111100000000111111111000000001111:Y6
>0110011110011000011001111001100001 10011:Y5
>101010110101010010101011010101001010101:Y4
>011111100000000111111000000001111110000:Y3
>100110011000011001100110000110011001100:Y2
>101001010100101010010101001010100101010:Y1

>111111111111111111111111111111111111111:D10
>000000000000000000000000000000000000000:D9
>000000000000000000000000000000000000000:D8
>000000000000000000000000000000000000000:D7
```

```
>0000000000000000000000000000000000000:D6

>0000000000000000000000000000000000000:D5

>0000000000000000000000000000000000000:D4

>0000000000000000000000000000000000000:D3

>0000000000000000000000000000000000000:D2

>0000000000000000000000000000000000000:D1


>111111:Y10

>011110:Y9

>100111:Y8

>101010:Y7

>111111:Y6

>111111:Y5

>111111:Y4

>111111:Y3

>111111:Y2

>111111:Y1


>111111:D10

>000000:D9

>000000:D8

>000000:D7

>000000:D6

>000000:D5
```

```
>000000:D4

>000000:D3

>000000:D2

>000000:D1
```

# APPENDIX K. MATLAB M-FILE CODE FOR 9-BIT ADC

## A. ADC TRANSFER CURVE

```
% adc15.m
% Thesis Design
% ADC Transfer Curve using ideal folding waveform output
% 15% Error Detection Decimation.

clear
clg

m7data
cpgrp15data
cpgrp8data
cpgrp11data

start=1;
stop=max(size(fold));

% Converting Modulus 7, 8, 11 Comparator Outputs to zeros & ones

t7a=Vmod715a(:,3:9)>1;
t7b=Vmod715b(:,3:9)>1;

t8=Vmod8(:,3:9)>1;

t11a=Vmod11a(:,3:7)>1;
t11b=Vmod11b(:,3:7)>1;

% Converting Modulus 7 Thermometer Output to decimal values
        mod7 = t7b(:,6) + t7b(:,4) + t7b(:,2) + t7a(:,7) + t7a(:,5) + t7a(:,3);

% Converting Modulus 8 Thermometer Output to decimal values
        mod8 = t8(:,7) + t8(:,6) + t8(:,5) + t8(:,4) + t8(:,3) + t8(:,2) + t8(:,1);

% Converting Modulus 11 Thermometer Output to decimal values
        mod11 = t11a(:,1) + t11a(:,2) + t11a(:,3) + t11a(:,4) + t11a(:,5) + t11b(:,1) + t11b(:,2) +
        t11b(:,3) + t11b(:,4) + t11b(:,5);


% Checking Parity of Modulus 7
% This consist of an Exclusive OR Configuration
% This is the first stage of the Parity Ckt
```

```
        p1 = ~((~t7b(:,7) & ~t7b(:,6)) | (t7b(:,7) & t7b(:,6)));
        p2 = ~((~t7b(:,4) & ~t7b(:,5)) | (t7b(:,4) & t7b(:,5)));
        p3 = ~((~t7b(:,2) & ~t7b(:,3)) | (t7b(:,2) & t7b(:,3)));
        p4 = ~((~t7b(:,1) & ~t7a(:,7)) | (t7b(:,1) & t7a(:,7)));
        p5 = ~((~t7a(:,5) & ~t7a(:,6)) | (t7a(:,5) & t7a(:,6)));
        p6 = ~((~t7a(:,3) & ~t7a(:,4)) | (t7a(:,3) & t7a(:,4)));
        p7 = ~((~t7a(:,1) & ~t7a(:,2)) | (t7a(:,1) & t7a(:,2)));

% This is the second stage of the circuit

        p8 = ~((~p6 & ~p5) | (p6 & p5));
        p9 = ~((~p3 & ~p4) | (p3 & p4));
        p10 = ~((~p1 & ~p2) | (p1 & p2));

% This is the third stage of the circuit

        p11 = ~((~p9 & ~p10) | (p9 & p10));
        p12 = ~((~p7 & ~p8) | (p7 & p8));

% This is the final stage

        Vpar = ~((~p11 & ~p12) | (p11 & p12));

% Converting Decimal Equivalent to Digital Output
start=1;
for i=start:stop
        if (mod7(i,1)==0 & mod8(i,1)==0 & mod11(i,1)==0)
         out(i,1)=0;
        elseif (mod7(i,1)==1 & mod8(i,1)==1 & mod11(i,1)==1)
         out(i,1)=1;
        elseif (mod7(i,1)==2 & mod8(i,1)==2 & mod11(i,1)==2)
         out(i,1)=2;
        elseif (mod7(i,1)==3 & mod8(i,1)==3 & mod11(i,1)==3)
         out(i,1)=3;
        elseif (mod7(i,1)==4 & mod8(i,1)==4 & mod11(i,1)==4)
         out(i,1)=4;
        elseif (mod7(i,1)==5 & mod8(i,1)==5 & mod11(i,1)==5)
         out(i,1)=5;
        elseif (mod7(i,1)==6 & mod8(i,1)==6 & mod11(i,1)==6)
         out(i,1)=6;
        elseif (mod7(i,1)==6 & mod8(i,1)==7 & mod11(i,1)==7)
         out(i,1)=7;
        elseif (mod7(i,1)==5 & mod8(i,1)==7 & mod11(i,1)==8)
         out(i,1)=8;
        elseif (mod7(i,1)==4 & mod8(i,1)==6 & mod11(i,1)==9)
```

```
  out(i,1)=9;
 elseif (mod7(i,1)==3 & mod8(i,1)==5 & mod11(i,1)==10)
  out(i,1)=10;
 elseif (mod7(i,1)==2 & mod8(i,1)==4 & mod11(i,1)==10)
  out(i,1)=11;
 elseif (mod7(i,1)==1 & mod8(i,1)==3 & mod11(i,1)==9)
  out(i,1)=12;
 elseif (mod7(i,1)==0 & mod8(i,1)==2 & mod11(i,1)==8)
  out(i,1)=13;
 elseif (mod7(i,1)==0 & mod8(i,1)==1 & mod11(i,1)==7)
  out(i,1)=14;
 elseif (mod7(i,1)==1 & mod8(i,1)==0 & mod11(i,1)==6)
  out(i,1)=15;
 elseif (mod7(i,1)==2 & mod8(i,1)==0 & mod11(i,1)==5)
  out(i,1)=16;
 elseif (mod7(i,1)==3 & mod8(i,1)==1 & mod11(i,1)==4)
  out(i,1)=17;
 elseif (mod7(i,1)==4 & mod8(i,1)==2 & mod11(i,1)==3)
  out(i,1)=18;
 elseif (mod7(i,1)==5 & mod8(i,1)==3 & mod11(i,1)==2)
  out(i,1)=19;
 elseif (mod7(i,1)==6 & mod8(i,1)==4 & mod11(i,1)==1)
  out(i,1)=20;
 elseif (mod7(i,1)==6 & mod8(i,1)==5 & mod11(i,1)==0)
  out(i,1)=21;
 elseif (mod7(i,1)==5 & mod8(i,1)==6 & mod11(i,1)==0)
  out(i,1)=22;
 elseif (mod7(i,1)==4 & mod8(i,1)==7 & mod11(i,1)==1)
  out(i,1)=23;
 elseif (mod7(i,1)==3 & mod8(i,1)==7 & mod11(i,1)==2)
  out(i,1)=24;
 elseif (mod7(i,1)==2 & mod8(i,1)==6 & mod11(i,1)==3)
  out(i,1)=25;
 elseif (mod7(i,1)==1 & mod8(i,1)==5 & mod11(i,1)==4)
  out(i,1)=26;
 elseif (mod7(i,1)==0 & mod8(i,1)==4 & mod11(i,1)==5)
  out(i,1)=27;
 elseif (mod7(i,1)==0 & mod8(i,1)==3 & mod11(i,1)==6)
  out(i,1)=28;
 elseif (mod7(i,1)==1 & mod8(i,1)==2 & mod11(i,1)==7)
  out(i,1)=29;
 elseif (mod7(i,1)==2 & mod8(i,1)==1 & mod11(i,1)==8)
  out(i,1)=30;
 elseif (mod7(i,1)==3 & mod8(i,1)==0 & mod11(i,1)==9)
  out(i,1)=31;
```

```
elseif (mod7(i,1)==4 & mod8(i,1)==0 & mod11(i,1)==10)
  out(i,1)=32;
elseif (mod7(i,1)==5 & mod8(i,1)==1 & mod11(i,1)==10)
  out(i,1)=33;
elseif (mod7(i,1)==6 & mod8(i,1)==2 & mod11(i,1)==9)
  out(i,1)=34;
elseif (mod7(i,1)==6 & mod8(i,1)==3 & mod11(i,1)==8)
  out(i,1)=35;
elseif (mod7(i,1)==5 & mod8(i,1)==4 & mod11(i,1)==7)
  out(i,1)=36;
elseif (mod7(i,1)==4 & mod8(i,1)==5 & mod11(i,1)==6)
  out(i,1)=37;
elseif (mod7(i,1)==3 & mod8(i,1)==6 & mod11(i,1)==5)
  out(i,1)=38;
elseif (mod7(i,1)==2 & mod8(i,1)==7 & mod11(i,1)==4)
  out(i,1)=39;
elseif (mod7(i,1)==1 & mod8(i,1)==7 & mod11(i,1)==3)
  out(i,1)=40;
elseif (mod7(i,1)==0 & mod8(i,1)==6 & mod11(i,1)==2)
  out(i,1)=41;
elseif (mod7(i,1)==0 & mod8(i,1)==5 & mod11(i,1)==1)
  out(i,1)=42;
elseif (mod7(i,1)==1 & mod8(i,1)==4 & mod11(i,1)==0)
  out(i,1)=43;
elseif (mod7(i,1)==2 & mod8(i,1)==3 & mod11(i,1)==0)
  out(i,1)=44;
elseif (mod7(i,1)==3 & mod8(i,1)==2 & mod11(i,1)==1)
  out(i,1)=45;
elseif (mod7(i,1)==4 & mod8(i,1)==1 & mod11(i,1)==2)
  out(i,1)=46;
elseif (mod7(i,1)==5 & mod8(i,1)==0 & mod11(i,1)==3)
  out(i,1)=47;
elseif (mod7(i,1)==6 & mod8(i,1)==0 & mod11(i,1)==4)
  out(i,1)=48;
elseif (mod7(i,1)==6 & mod8(i,1)==1 & mod11(i,1)==5)
  out(i,1)=49;
elseif (mod7(i,1)==5 & mod8(i,1)==2 & mod11(i,1)==6)
  out(i,1)=50;
elseif (mod7(i,1)==4 & mod8(i,1)==3 & mod11(i,1)==7)
  out(i,1)=51;
elseif (mod7(i,1)==3 & mod8(i,1)==4 & mod11(i,1)==8)
  out(i,1)=52;
elseif (mod7(i,1)==2 & mod8(i,1)==5 & mod11(i,1)==9)
  out(i,1)=53;
elseif (mod7(i,1)==1 & mod8(i,1)==6 & mod11(i,1)==10)
```

```
 out(i,1)=54;
elseif (mod7(i,1)==0 & mod8(i,1)==7 & mod11(i,1)==10)
 out(i,1)=55;
elseif (mod7(i,1)==0 & mod8(i,1)==7 & mod11(i,1)==9)
 out(i,1)=56;
elseif (mod7(i,1)==1 & mod8(i,1)==6 & mod11(i,1)==8)
 out(i,1)=57;
elseif (mod7(i,1)==2 & mod8(i,1)==5 & mod11(i,1)==7)
 out(i,1)=58;
elseif (mod7(i,1)==3 & mod8(i,1)==4 & mod11(i,1)==6)
 out(i,1)=59;
elseif (mod7(i,1)==4 & mod8(i,1)==3 & mod11(i,1)==5)
 out(i,1)=60;
elseif (mod7(i,1)==5 & mod8(i,1)==2 & mod11(i,1)==4)
 out(i,1)=61;
elseif (mod7(i,1)==6 & mod8(i,1)==1 & mod11(i,1)==3)
 out(i,1)=62;
elseif (mod7(i,1)==6 & mod8(i,1)==0 & mod11(i,1)==2)
 out(i,1)=63;
elseif (mod7(i,1)==5 & mod8(i,1)==0 & mod11(i,1)==1)
 out(i,1)=64;
elseif (mod7(i,1)==4 & mod8(i,1)==1 & mod11(i,1)==0)
 out(i,1)=65;
elseif (mod7(i,1)==3 & mod8(i,1)==2 & mod11(i,1)==0)
 out(i,1)=66;
elseif (mod7(i,1)==2 & mod8(i,1)==3 & mod11(i,1)==1)
 out(i,1)=67;
elseif (mod7(i,1)==1 & mod8(i,1)==4 & mod11(i,1)==2)
 out(i,1)=68;
elseif (mod7(i,1)==0 & mod8(i,1)==5 & mod11(i,1)==3)
 out(i,1)=69;
elseif (mod7(i,1)==0 & mod8(i,1)==6 & mod11(i,1)==4)
 out(i,1)=70;
elseif (mod7(i,1)==1 & mod8(i,1)==7 & mod11(i,1)==5)
 out(i,1)=71;
elseif (mod7(i,1)==2 & mod8(i,1)==7 & mod11(i,1)==6)
 out(i,1)=72;
elseif (mod7(i,1)==3 & mod8(i,1)==6 & mod11(i,1)==7)
 out(i,1)=73;
elseif (mod7(i,1)==4 & mod8(i,1)==5 & mod11(i,1)==8)
 out(i,1)=74;
elseif (mod7(i,1)==5 & mod8(i,1)==4 & mod11(i,1)==9)
 out(i,1)=75;
elseif (mod7(i,1)==6 & mod8(i,1)==3 & mod11(i,1)==10)
 out(i,1)=76;
```

```
elseif (mod7(i,1)==6 & mod8(i,1)==2 & mod11(i,1)==10)
 out(i,1)=77;
elseif (mod7(i,1)==5 & mod8(i,1)==1 & mod11(i,1)==9)
 out(i,1)=78;
elseif (mod7(i,1)==4 & mod8(i,1)==0 & mod11(i,1)==8)
 out(i,1)=79;
elseif (mod7(i,1)==3 & mod8(i,1)==0 & mod11(i,1)==7)
 out(i,1)=80;
elseif (mod7(i,1)==2 & mod8(i,1)==1 & mod11(i,1)==6)
 out(i,1)=81;
elseif (mod7(i,1)==1 & mod8(i,1)==2 & mod11(i,1)==5)
 out(i,1)=82;
elseif (mod7(i,1)==0 & mod8(i,1)==3 & mod11(i,1)==4)
 out(i,1)=83;
elseif (mod7(i,1)==0 & mod8(i,1)==4 & mod11(i,1)==3)
 out(i,1)=84;
elseif (mod7(i,1)==1 & mod8(i,1)==5 & mod11(i,1)==2)
 out(i,1)=85;
elseif (mod7(i,1)==2 & mod8(i,1)==6 & mod11(i,1)==1)
 out(i,1)=86;
elseif (mod7(i,1)==3 & mod8(i,1)==7 & mod11(i,1)==0)
 out(i,1)=87;
elseif (mod7(i,1)==4 & mod8(i,1)==7 & mod11(i,1)==0)
 out(i,1)=88;
elseif (mod7(i,1)==5 & mod8(i,1)==6 & mod11(i,1)==1)
 out(i,1)=89;
elseif (mod7(i,1)==6 & mod8(i,1)==5 & mod11(i,1)==2)
 out(i,1)=90;
elseif (mod7(i,1)==6 & mod8(i,1)==4 & mod11(i,1)==3)
 out(i,1)=91;
elseif (mod7(i,1)==5 & mod8(i,1)==3 & mod11(i,1)==4)
 out(i,1)=92;
elseif (mod7(i,1)==4 & mod8(i,1)==2 & mod11(i,1)==5)
 out(i,1)=93;
elseif (mod7(i,1)==3 & mod8(i,1)==1 & mod11(i,1)==6)
 out(i,1)=94;
elseif (mod7(i,1)==2 & mod8(i,1)==0 & mod11(i,1)==7)
 out(i,1)=95;
elseif (mod7(i,1)==1 & mod8(i,1)==0 & mod11(i,1)==8)
 out(i,1)=96;
elseif (mod7(i,1)==0 & mod8(i,1)==1 & mod11(i,1)==9)
 out(i,1)=97;
elseif (mod7(i,1)==0 & mod8(i,1)==2 & mod11(i,1)==10)
 out(i,1)=98;
elseif (mod7(i,1)==1 & mod8(i,1)==3 & mod11(i,1)==10)
```

```
 out(i,1)=99;
elseif (mod7(i,1)==2 & mod8(i,1)==4 & mod11(i,1)==9)
 out(i,1)=100;
elseif (mod7(i,1)==3 & mod8(i,1)==5 & mod11(i,1)==8)
 out(i,1)=101;
elseif (mod7(i,1)==4 & mod8(i,1)==6 & mod11(i,1)==7)
 out(i,1)=102;
elseif (mod7(i,1)==5 & mod8(i,1)==7 & mod11(i,1)==6)
 out(i,1)=103;
elseif (mod7(i,1)==6 & mod8(i,1)==7 & mod11(i,1)==5)
 out(i,1)=104;
elseif (mod7(i,1)==6 & mod8(i,1)==6 & mod11(i,1)==4)
 out(i,1)=105;
elseif (mod7(i,1)==5 & mod8(i,1)==5 & mod11(i,1)==3)
 out(i,1)=106;
elseif (mod7(i,1)==4 & mod8(i,1)==4 & mod11(i,1)==2)
 out(i,1)=107;
elseif (mod7(i,1)==3 & mod8(i,1)==3 & mod11(i,1)==1)
 out(i,1)=108;
elseif (mod7(i,1)==2 & mod8(i,1)==2 & mod11(i,1)==0)
 out(i,1)=109;
elseif (mod7(i,1)==1 & mod8(i,1)==1 & mod11(i,1)==0)
 out(i,1)=110;
elseif (mod7(i,1)==0 & mod8(i,1)==0 & mod11(i,1)==1)
 out(i,1)=111;
elseif (mod7(i,1)==0 & mod8(i,1)==0 & mod11(i,1)==2)
 out(i,1)=112;
elseif (mod7(i,1)==1 & mod8(i,1)==1 & mod11(i,1)==3)
 out(i,1)=113;
elseif (mod7(i,1)==2 & mod8(i,1)==2 & mod11(i,1)==4)
 out(i,1)=114;
elseif (mod7(i,1)==3 & mod8(i,1)==3 & mod11(i,1)==5)
 out(i,1)=115;
elseif (mod7(i,1)==4 & mod8(i,1)==4 & mod11(i,1)==6)
 out(i,1)=116;
elseif (mod7(i,1)==5 & mod8(i,1)==5 & mod11(i,1)==7)
 out(i,1)=117;
elseif (mod7(i,1)==6 & mod8(i,1)==6 & mod11(i,1)==8)
 out(i,1)=118;
elseif (mod7(i,1)==6 & mod8(i,1)==7 & mod11(i,1)==9)
 out(i,1)=119;
elseif (mod7(i,1)==5 & mod8(i,1)==7 & mod11(i,1)==10)
 out(i,1)=120;
elseif (mod7(i,1)==4 & mod8(i,1)==6 & mod11(i,1)==10)
 out(i,1)=121;
```

```
elseif (mod7(i,1)==3 & mod8(i,1)==5 & mod11(i,1)==9)
  out(i,1)=122;
elseif (mod7(i,1)==2 & mod8(i,1)==4 & mod11(i,1)==8)
  out(i,1)=123;
elseif (mod7(i,1)==1 & mod8(i,1)==3 & mod11(i,1)==7)
  out(i,1)=124;
elseif (mod7(i,1)==0 & mod8(i,1)==2 & mod11(i,1)==6)
  out(i,1)=125;
elseif (mod7(i,1)==0 & mod8(i,1)==1 & mod11(i,1)==5)
  out(i,1)=126;
elseif (mod7(i,1)==1 & mod8(i,1)==0 & mod11(i,1)==4)
  out(i,1)=127;
elseif (mod7(i,1)==2 & mod8(i,1)==0 & mod11(i,1)==3)
  out(i,1)=128;
elseif (mod7(i,1)==3 & mod8(i,1)==1 & mod11(i,1)==2)
  out(i,1)=129;
elseif (mod7(i,1)==4 & mod8(i,1)==2 & mod11(i,1)==1)
  out(i,1)=130;
elseif (mod7(i,1)==5 & mod8(i,1)==3 & mod11(i,1)==0)
  out(i,1)=131;
elseif (mod7(i,1)==6 & mod8(i,1)==4 & mod11(i,1)==0)
  out(i,1)=132;
elseif (mod7(i,1)==6 & mod8(i,1)==5 & mod11(i,1)==1)
  out(i,1)=133;
elseif (mod7(i,1)==5 & mod8(i,1)==6 & mod11(i,1)==2)
  out(i,1)=134;
elseif (mod7(i,1)==4 & mod8(i,1)==7 & mod11(i,1)==3)
  out(i,1)=135;
elseif (mod7(i,1)==3 & mod8(i,1)==7 & mod11(i,1)==4)
  out(i,1)=136;
elseif (mod7(i,1)==2 & mod8(i,1)==6 & mod11(i,1)==5)
  out(i,1)=137;
elseif (mod7(i,1)==1 & mod8(i,1)==5 & mod11(i,1)==6)
  out(i,1)=138;
elseif (mod7(i,1)==0 & mod8(i,1)==4 & mod11(i,1)==7)
  out(i,1)=139;
elseif (mod7(i,1)==0 & mod8(i,1)==3 & mod11(i,1)==8)
  out(i,1)=140;
elseif (mod7(i,1)==1 & mod8(i,1)==2 & mod11(i,1)==9)
  out(i,1)=141;
elseif (mod7(i,1)==2 & mod8(i,1)==1 & mod11(i,1)==10)
  out(i,1)=142;
elseif (mod7(i,1)==3 & mod8(i,1)==0 & mod11(i,1)==10)
  out(i,1)=143;
elseif (mod7(i,1)==4 & mod8(i,1)==0 & mod11(i,1)==9)
```

```
 out(i,1)=144;
elseif (mod7(i,1)==5 & mod8(i,1)==1 & mod11(i,1)==8)
 out(i,1)=145;
elseif (mod7(i,1)==6 & mod8(i,1)==2 & mod11(i,1)==7)
 out(i,1)=146;
elseif (mod7(i,1)==6 & mod8(i,1)==3 & mod11(i,1)==6)
 out(i,1)=147;
elseif (mod7(i,1)==5 & mod8(i,1)==4 & mod11(i,1)==5)
 out(i,1)=148;
elseif (mod7(i,1)==4 & mod8(i,1)==5 & mod11(i,1)==4)
 out(i,1)=149;
elseif (mod7(i,1)==3 & mod8(i,1)==6 & mod11(i,1)==3)
 out(i,1)=150;
elseif (mod7(i,1)==2 & mod8(i,1)==7 & mod11(i,1)==2)
 out(i,1)=151;
elseif (mod7(i,1)==1 & mod8(i,1)==7 & mod11(i,1)==1)
 out(i,1)=152;
elseif (mod7(i,1)==0 & mod8(i,1)==6 & mod11(i,1)==0)
 out(i,1)=153;
elseif (mod7(i,1)==0 & mod8(i,1)==5 & mod11(i,1)==0)
 out(i,1)=154;
elseif (mod7(i,1)==1 & mod8(i,1)==4 & mod11(i,1)==1)
 out(i,1)=155;
elseif (mod7(i,1)==2 & mod8(i,1)==3 & mod11(i,1)==2)
 out(i,1)=156;
elseif (mod7(i,1)==3 & mod8(i,1)==2 & mod11(i,1)==3)
 out(i,1)=157;
elseif (mod7(i,1)==4 & mod8(i,1)==1 & mod11(i,1)==4)
 out(i,1)=158;
elseif (mod7(i,1)==5 & mod8(i,1)==0 & mod11(i,1)==5)
 out(i,1)=159;
elseif (mod7(i,1)==6 & mod8(i,1)==0 & mod11(i,1)==6)
 out(i,1)=160;
elseif (mod7(i,1)==6 & mod8(i,1)==1 & mod11(i,1)==7)
 out(i,1)=161;
elseif (mod7(i,1)==5 & mod8(i,1)==2 & mod11(i,1)==8)
 out(i,1)=162;
elseif (mod7(i,1)==4 & mod8(i,1)==3 & mod11(i,1)==9)
 out(i,1)=163;
elseif (mod7(i,1)==3 & mod8(i,1)==4 & mod11(i,1)==10)
 out(i,1)=164;
elseif (mod7(i,1)==2 & mod8(i,1)==5 & mod11(i,1)==10)
 out(i,1)=165;
elseif (mod7(i,1)==1 & mod8(i,1)==6 & mod11(i,1)==9)
 out(i,1)=166;
```

```
elseif (mod7(i,1)==0 & mod8(i,1)==7 & mod11(i,1)==8)
 out(i,1)=167;
elseif (mod7(i,1)==0 & mod8(i,1)==7 & mod11(i,1)==7)
 out(i,1)=168;
elseif (mod7(i,1)==1 & mod8(i,1)==6 & mod11(i,1)==6)
 out(i,1)=169;
elseif (mod7(i,1)==2 & mod8(i,1)==5 & mod11(i,1)==5)
 out(i,1)=170;
elseif (mod7(i,1)==3 & mod8(i,1)==4 & mod11(i,1)==4)
 out(i,1)=171;
elseif (mod7(i,1)==4 & mod8(i,1)==3 & mod11(i,1)==3)
 out(i,1)=172;
elseif (mod7(i,1)==5 & mod8(i,1)==2 & mod11(i,1)==2)
 out(i,1)=173;
elseif (mod7(i,1)==6 & mod8(i,1)==1 & mod11(i,1)==1)
 out(i,1)=174;
elseif (mod7(i,1)==6 & mod8(i,1)==0 & mod11(i,1)==0)
 out(i,1)=175;
elseif (mod7(i,1)==5 & mod8(i,1)==0 & mod11(i,1)==0)
 out(i,1)=176;
elseif (mod7(i,1)==4 & mod8(i,1)==1 & mod11(i,1)==1)
 out(i,1)=177;
elseif (mod7(i,1)==3 & mod8(i,1)==2 & mod11(i,1)==2)
 out(i,1)=178;
elseif (mod7(i,1)==2 & mod8(i,1)==3 & mod11(i,1)==3)
 out(i,1)=179;
elseif (mod7(i,1)==1 & mod8(i,1)==4 & mod11(i,1)==4)
 out(i,1)=180;
elseif (mod7(i,1)==0 & mod8(i,1)==5 & mod11(i,1)==5)
 out(i,1)=181;
elseif (mod7(i,1)==0 & mod8(i,1)==6 & mod11(i,1)==6)
 out(i,1)=182;
elseif (mod7(i,1)==1 & mod8(i,1)==7 & mod11(i,1)==7)
 out(i,1)=183;
elseif (mod7(i,1)==2 & mod8(i,1)==7 & mod11(i,1)==8)
 out(i,1)=184;
elseif (mod7(i,1)==3 & mod8(i,1)==6 & mod11(i,1)==9)
 out(i,1)=185;
elseif (mod7(i,1)==4 & mod8(i,1)==5 & mod11(i,1)==10)
 out(i,1)=186;
elseif (mod7(i,1)==5 & mod8(i,1)==4 & mod11(i,1)==10)
 out(i,1)=187;
elseif (mod7(i,1)==6 & mod8(i,1)==3 & mod11(i,1)==9)
 out(i,1)=188;
elseif (mod7(i,1)==6 & mod8(i,1)==2 & mod11(i,1)==8)
```

```
          out(i,1)=189;
         elseif (mod7(i,1)==5 & mod8(i,1)==1 & mod11(i,1)==7)
          out(i,1)=190;
         elseif (mod7(i,1)==4 & mod8(i,1)==0 & mod11(i,1)==6)
          out(i,1)=191;
         elseif (mod7(i,1)==3 & mod8(i,1)==0 & mod11(i,1)==5)
          out(i,1)=192;
         elseif (mod7(i,1)==2 & mod8(i,1)==1 & mod11(i,1)==4)
          out(i,1)=193;
         elseif (mod7(i,1)==1 & mod8(i,1)==2 & mod11(i,1)==3)
          out(i,1)=194;
         elseif (mod7(i,1)==0 & mod8(i,1)==3 & mod11(i,1)==2)
          out(i,1)=195;
         elseif (mod7(i,1)==0 & mod8(i,1)==4 & mod11(i,1)==1)
          out(i,1)=196;
         elseif (mod7(i,1)==1 & mod8(i,1)==5 & mod11(i,1)==0)
          out(i,1)=197;
         elseif (mod7(i,1)==2 & mod8(i,1)==6 & mod11(i,1)==0)
          out(i,1)=198;
         elseif (mod7(i,1)==3 & mod8(i,1)==7 & mod11(i,1)==1)
          out(i,1)=199;
         elseif (mod7(i,1)==4 & mod8(i,1)==7 & mod11(i,1)==2)
          out(i,1)=200;
         else
          out(i,1)=800; %Error Code for output
         end
end


% Check the parity, if even then discard value and hold output at
% last know good value

temp=0;
stop=max(size(out));
for i=start:stop
         if (Vpar(i))      % parity bit=1, keep
          errorcor(i)=out(i);
          temp=out(i);
         else              % parity bit=0, discard
          errorcor(i)=temp;
         end
end

ec=errorcor';
```

```
% Error Counting
% Compute the ideal line corresponding to the transfer curve

  p = polyfit(fold([start,stop],2),ec([start,stop],1),1);
  yfitted = polyval(p,fold(:,2));

%Looking for errors>1 at every step

  error = abs(ec(:,1)-yfitted);

%find returns the location of all values that meet the given criteria.

  f = find(error>1.5);

%errorcnt will contain the number of errors

  errorcnt15 = length(f);

%finding linearity errors

linerror15=(ec(:,1)-yfitted);
linearerror=max(linerror15)*100/29.64;


% Plot the error corrected output
plot(fold(:,2), ec(:,1))
title('9-Bit ADC Transfer Curve with 15% Decimation')
ylabel('ADC Decimal Output Code')
xlabel('ADC Input Voltage')
grid
axis([0 5.5 0 200])
```

## B. LINEARITY ERROR

```
% M-file to plot linearity error of a 9-bit ADC

theo=[];
for x=1:200
 for i=1:30
  theo(x,i) = x-1;
 end
end
```

```
for a=1:6000
  xaxes(a)=a-1;
end

tmpa=.988e-3.*xaxes;

tmpx=reshape(tmpa,30,200);
tmpx=tmpx';

hold on
for i=1:200
 plot(tmpx(i,:),theo(i,:))
 hold on
end

hold on
plot(fold(:,2),ec(:,1))
title('Ideal vs. Actual ADC Output')
ylabel('ADC Decimal Output Code')
xlabel('ADC Input Voltage')
grid
axis([0 0.2 0 6])
axis('square')
```

# LIST OF REFERENCES

1. Coughlin, R.F. and Driscoll, F.F., *Operational Amplifiers & Linear Integrated Circuits,* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1991.

2. Pace, P. E., Ramamoorthy, P. A., and Styer, D., "A Preprocessing Architecture for Resolution Enhancement in High Speed Analog-to-Digital Converters," *IEEE Transactions on Circuits and Systems*, vol. 41, pp. 373-379, 1994.

3. Van Valburg, J. and Van de Plassche, R.J., "An 8-b 650-MHz Folding ADC," *IEEE Journal of Solid-State Circuits,* Vol. 27, pp. 1662-1666, 1992.

4. Van de Grift, R.E.J. and Van de Plassche, R.J., "A Monolithic 8-Bit Video A/D Converter," *IEEE Journal of Solid-State Circuits,* Vol. 19, pp. 374-378, 1984.

5. Van de Plassche, R.J. and Baltus, P., "An 8-bit 100-MHz Full-Nyquist Analog-to-Digital Converter," *IEEE Journal of Solid-State Circuits,* Vol. 23, pp. 1334-1344, 1988.

6. Van de Grift, R.E.J., Rutten, I.W.J.M., and Van der Veen, M., "An 8-bit Video ADC Incorporating Folding and Interpolation Techniques," *IEEE Journal of Solid-State Circuits,* Vol. 22, pp. 944-953, 1987.

7. Esparza, J. A., *An Analog Preprocessing Architecture for High-Speed Analog-to-Digital Conversion*, Master Thesis, Naval Postgraduate School, Monterey, CA, December 1993.

8. J. E. Shockley, *Introduction to Number Theory,* Holt, Rhinehart and Winston, Inc., New York, NY, 1967.

9. Pace, P. E. and Styer, D., "High Resolution Encoding Process for an Integrated Optical Analog-to-Digital Converter," *Optical Engineering*, Vol. 33, pp. 2638-2645, 1994.

10. Pace, P. E., Schafer, J. L., and Styer, D., "Optimum Analog Preprocessing for Folding ADCs," *IEEE Transactions on Circuits and Systems-II*, Submitted September 1994.

11. Computer Science Division, University of California at Berkeley, *Berkeley CAD Tools User's Manual*, University of California at Berkeley, 1986.

12. Meta-Software, Inc., *HSPICE User's Manual - HSPICE Version H92*, Meta-Software, Inc., Campbell, CA, 1992.

13. Math Works Inc., *386-MATLAB User's Guide,* Math Works Inc., Natick, MA, 1991.

14. MicroSim Corp., *The Design Center: Schematic Capture User's Guide - Version 5.3*, MicroSim Corporation, Irvine, CA, 1993.

15. Carr, R. D., *Analog Preprocessing in a SNS 2μ Low-Noise CMOS Folding ADC*, Master Thesis, Naval Postgraduate School, Monterey, CA, December 1994.

16. West, N.H. and Eshraghian, K., *Principles of CMOS VLSI Design,* Addison-Wesley Inc., Reading, MA, 1993.

# BIBLIOGRAPHY

Allen, P. E., and Holdberg, D. R., *CMOS Analog Circuit Design*, Holt, Rinehart, and Winston, Inc., New York, NY, 1987.

Szabo, N. S., and Tanaka, R. I., *Residue Arithmetic and Its Application to Computer Technology*, McGraw-Hill, Inc., New York, NY, 1967.

Weste, N. H. E., Eshraghian, K., *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley, Inc., Reading, MA, 1993.

## INITIAL DISTRIBUTION LIST

|  |  |  |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 52<br>Naval Postgraduate School<br>Monterey, California 93943-5101 | 2 |
| 3. | Chairman, Code EC<br>Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, California 93943-5121 | 1 |
| 4. | Professor P. Pace, Code EC/Pc<br>Naval Postgraduate School<br>Monterey, California 93943-5121 | 3 |
| 5. | Professor D. Fouts, Code EC/Fs<br>Naval Postgraduate School<br>Monterey, California 93943-5121 | 1 |
| 6. | Instructor R. Borchardt, Code EC/Bt<br>Naval Postgraduate School<br>Monterey, California 93943-5121 | 1 |
| 7. | LT Jeffrey L. Schafer, USN<br>218 W. Grove Street<br>Greenville, Michigan 48838 | 2 |
| 8. | LCDR Richard D. Carr, USN<br>472 Springfield Road<br>Mount Pleasance, South Carolina 29464 | 1 |